APPENDIX

Express Mail No. EL360386749US

VSI Alliance™
Virtual Component Interface Standard
(OCB 2 1.0)
On-Chip Bus Development Working Group
Standard 2 Version 1.0

Working Revision (2/2/2000)

Members of the Development Working Group

ARM	Cadence
Co-design	Easics
ECSI	Fujitsu
GDA Tech	Hitachi
HP	LSI Logic
Lucent	Nokia
OKI	Palmchip
Philips	Phoenix
Siemens	Sonics
ST Microelectronics	Synopsys
Toshiba	Xilinx

Active Contributors

Bruce Mathewson ..... ARM
Larry Cooke* (Ex Chair)..... Cadence
Mani Gopalakrishnan*..... Fujitsu
J. Sukarno Mertoguno* (co Chair) ..... Fujitsu
Prakash Bare* ..... GDA Tech
Janet Wedgwood..... Lockheed Martin
Rick Born*..... LSI
Anssi Haverinen* (Chair)..... Nokia
Frits Zandveld*..... Philips
Amjad Qureshi ..... Phoenix
David Jennings ..... Siemens
John Carey*..... ST Microelectronics
Graham Matthew*(Ex Chair)..... ST Microelectronics
Mike Meyer.....Sonics
Drew Wingard*..... Sonics
Greg Tanaka ..... Synopsys
Frank Vahid ..... Univ Calif
Prof. Ganesh Gopalakrishnan ..... Univ Utah
Glen Baxter ..... Xilinx

Other Contributors

John Cornish .....ARM
Larry Rosenberg.....Cadence
Peter Flake .....Co-design
Tony Gore .....ECSI
Gerry Caracciolo.....Eigen Tek
Glen Vinogradov.....independent
Yuji Hatano.....Hitachi
Osamu Yamashiro .....Hitachi
Won Rhee.....HP
Alex Kurosawa .....Sonics
Charles Minter .....Toshiba
Joseph Hassoun.....Xilinx

* - Major contributors

TABLE OF CONTENTS

1. Virtual Component Interface (VCI) Background

This standard is the result of the work of the On-Chip Bus (OCB) Development Working Group (DWG) of the Virtual Socket Interface Alliance (VSIA). The charter defined for the OCB DWG was to define on-chip bus -related specifications for the design, integration, and test of multiple functional blocks on a single piece of silicon.

Two generic audiences are targeted: Providers of functional blocks and Users or Integrators of these blocks. For the Providers of functional blocks, the standard defines the protocols and interfaces their Users require for effective reuse of the blocks in an integrated product design. For the Users of functional blocks in compliance with this standard, it depicts the information these Users will need to properly evaluate, integrate, and verify one or more functional blocks in a design.

The overall objective is to obtain a general interface, such that Intellectual Property (IP), in the shape of Virtual Components (VC) of any origin, can be connected to Systems-on-Chips (SoC) of any chip Integrator. In this manner, VCs are not limited to one-time usage by their designers. They can be **re-used** over and over. Such re-use of VCs would also apply to VC Providers or to (external) System Integrators.

Early in the existence of this DWG it became clear that picking an existing bus or defining a new one would not be the right way to go. First of all, System Integrators will stick to their own bus for a relatively long time, if only to allow for connection of their existing VCs without modification. Connecting new VCs via a bus-to-bus bridge is expensive in timing and in silicon footprint. Furthermore, all existing buses are rather good for their particular usage. Inventing yet another one does not make much sense. Thus there would be little chance of such a bus being accepted.

For these reasons, the working group decided to define an **interface** rather than a bus. This interface can then be used as a point to point connection if one is needed and additionally as an interface to a bus connector if that is what is desired.

1.1 Goals

The following were considered primary goals in defining the VCI.

1. Must enable maximum portability of a VC
   - *VCI-compliant VCs should inter-operate with OCBs of varying protocols and performance levels.*
   - *VCI should not dictate the integration methodology, i.e. VCI can be used as a point-to-point interface without an OCB, and can be connected to different OCBs with automated methods, or with hand-crafted wrappers.*

2. Should not require modification of VCI-compliant VCs in order to connect to a different VCI-compliant VC or OCB.
   - *Some combinations of VCI-compliant VCs and OCBs may result in reduced features or performance, but must still function correctly and reliably.*

3. Simple and efficient to implement with clear and easy to understand protocol
   - *Necessary for wide acceptance.*

4. Design a compatible family of VC Interfaces (i.e. Advanced VCI, Basic VCI, and Peripheral VCI)
   - *This enhances inter-operability choices for the System Chip designer, and design opportunities for the VC designer.*

5. Protocol must be fully defined

- This includes full enumeration of possible interactions together with statements on allowed / disallowed behavior. However, it is essential for wide adoption that there be some mechanism for extending *the protocol.*

6. Few optional signals

- Minimize the number of truly optional signals to minimize the complexity of VCI compliance checking. However, signals such as data and address lines and supported transfer widths may be parameterizable and/or scalable.

1.2 Key Assumptions

This section describes the assumptions that were made in defining the VCI.

1. Initiator/Target connections are point-to-point and unidirectional.

- Both multiplexed and tri-state OCBs can be supported by allowing the OCB wrappers to implement the OCB transceivers. Separate unidirectional nets are simpler to handle, and this circumvents the requirement for arbitration in the VCI protocol.

2. Initiator can only present requests; Target can only respond

- If a VC needs both capabilities, implement parallel Initiator and Target interfaces.

- A combined (peer-to-peer) protocol is much more complex to define and implement (due to interface arbitration requirements).
- Some OCB architectures can take advantage of the separate initiator and target interfaces by connecting the VC Initiator interface to a System OCB whereas the (separate) VC Target interface might connect to a Peripheral or Configuration OCB (a bus for loading device configurations).
- Finally, the parallel interface is simpler to model and offers possible performance advantages using simultaneous transfers.

- Limited fundamental Read / Write requests.

- Peripheral VCI is limited to read and write
- Basic VCI includes Nop and read lock.

3. Address / Data widths are determined by VC requirements

- OCB Target's Initiator should scale its address / data widths to match Target

4. The standard should ensure that any required data and address storage in the wrappers is minimal.

- Anything more than minimal timing overhead is deemed unacceptable and would negatively impact acceptance.

5. Clock domain crossing will not be visible at the interface

- Interface is fully synchronous. Clock domain crossing is considered a design guideline for wrapper or VC implementation.

1.3 Document Conventions

b == number of bytes in a cell
n == most significant bit of address field
m == least significant bit of cell-address
k == Number of bits in the packet length field
q == Number of bits in the packet chain length field
E == Number of bits in the error extension field

Timing constraints:

Early: Signal is Valid within 20% of the clock cycle from the rise of the Clock signal

Middle: Signal is Valid within 50% of the clock cycle from the rise of the Clock signal
Late : Signal is Valid within 80% of the clock cycle from the rise of the Clock signal

1.4 Document Organization

The Standard begins with defining the basic characteristics of the VCI. The different-complexity interfaces are described in detail next, starting with the simplest one, the Peripheral VCI. The PVCI and BVCI sections are designed so that they can be read independently, i.e. if a reader wants to know about to use the PVCI, he does not need to be familiar with the BVCI. After the interface descriptions, some design guidelines are given. VCI Transaction Language section will be included in the standard version 2 2.0.

2. VCI Characteristics

2.1 VCI Definition

The Virtual Component Interface or VCI is an interface rather than a bus. Thus the VCI specifies:

1. A requests-response protocol.
2. A protocol for the transfer of requests and responses.
3. The contents and coding of these requests and responses.

The VCI does not touch areas as bus allocation schemes, competing for a bus, etc.

There are three complexity levels for the VCI: Advanced VCI, Basic VCI, and Peripheral VCI (AVCI, BVCI, and PVCI respectively). The current version of the VCI standard defines the Basic VCI and the Peripheral VCI. The Advanced VCI will be added later. The Basic VCI defines an interface, which is suitable for most applications. It has a powerful, but not overly complex protocol. The Advanced VCI will add more sophisticated features, such as treads, to support high performance applications. The Peripheral VCI provides a simple, easily implementable interface for applications, which do not need all the features of the Basic VCI. The Peripheral VCI is a subset, and the Advanced VCI is a superset of the Basic VCI, and all those interfaces are designed to be compatible with each other.

2.2 Point to Point Usage

As an interface, the VCI can be used as a point to point connection between two units called the Initiator and the Target, in which the Initiator issues a request and the Target responds. The VCI can also be used as the interface to a "wrapper", a connection to a bus. This is how the VCI allows the Virtual Component to be connected to any bus (See section 2.4). Basic point to point usage is depicted below in Figure 1.

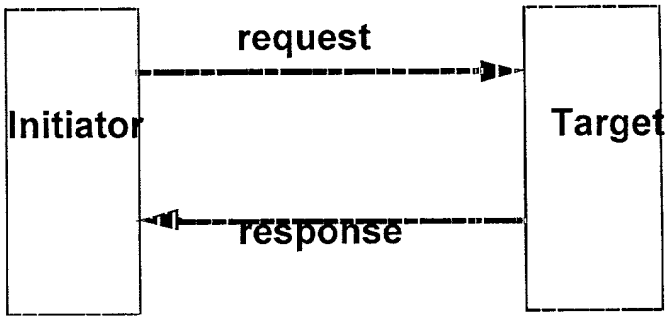

Figure 1. VCI Is a Point to Point Connection

This interface is very simple indeed while protocol and coding are concerned. Nevertheless, the interface contains many sophisticated features, as will be explained in further chapters of this standard. Its simplicity is the reason for the small footprint, and the high bandwidth.

2.3 Split Protocol

Basic VCI and Advanced VCI make use of a "split protocol". That is, the timing of the request and the response are fully separate. The Initiator can issue as many requests as it sees fit, without waiting for the response. The protocol does not prescribe any connection between issuing of requests and arrival of the corresponding responses. The only thing specified is that the order of responses corresponds to the order of requests.

In the Advanced VCI, requests may be tagged with identifiers, which allow such requests and request threads to be interleaved and even allows responses to arrive in a different order. Responses shall bear the same tags issued with the corresponding requests, such that the relation can be restored upon the reception of a response. Additional details will be provided as the Advanced VCI is standardized.

In the Peripheral VCI, no split protocol is used, and each request must be followed by a response, before the Initiator can issue a new request. This simplification is for supporting simple peripheral buses.

2.4 VCI Usage with a Bus

The VCI can well be used as the interface to a "wrapper", which means a connection to a bus. This is how the VCI allows the VC to be connected to any bus. An Initiator is connected to that bus via a bus initiator wrapper. A Target is connected to that bus via a bus target wrapper. Once the wrappers for the bus have been designed, any VC can be connected to that bus, as depicted below in Figure 2. The wrapper and the VCI-boxes (the non-shaded area) in the picture together correspond to the traditional bus interface within both the initiator and the target. The whole interface will, again, function as a point-to-point connection. Of course, the use of VCI does not prevent using native bus components at the same time. In fact, the presented use of VCI is completely transparent to the bus system.

It is intended that OCB suppliers will provide VCI wrappers to their proprietary bus, or EDA vendors will provide tools to create such wrappers automatically. This will free the IP Provider from having to understand the details of many buses.

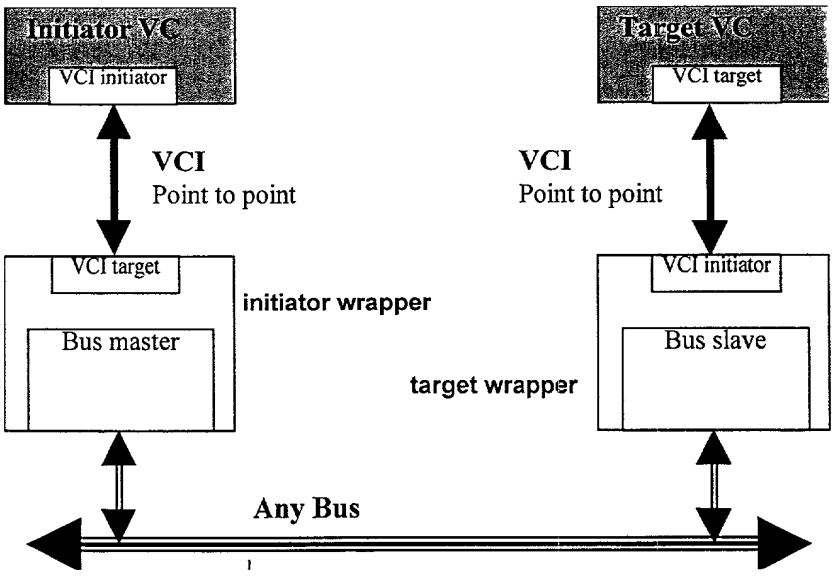

Figure 2. Two VCI Connections Used to Realize a Bus Connection

2.5 Notes

1. The "Initiator" and "Target" are simpler than their equivalents directly connected to a bus. The VCI initiator and the VCI target are much simpler than the bus initiator and target interfaces. The bus interface, though, is not removed. It is in the wrappers.

2. The wrapper contains not much more than the bus interface, which is needed anyway when connecting (no matter what) to the bus and to the VC interface block.

3. The two blocks marked "vci initiator" and the two blocks marked "vci target" in Figure 2 are very simple. They do not add much to the total footprint or timing. This is especially so, as some logic is needed quite often anyway to connect the actual functionality of Initiator and Target to their bus interfaces. *It is merely the vci_initiator – vci_target interface that is standardized in the VCI.*

3. Peripheral VCI

3.1 Overview

3.1.1 Scope

This section defines the first member of the VCI family, the Peripheral Virtual Component Interface (PVCI), which can be used in conjunction with peripheral on-chip buses as defined in the VSIA OCB Specification 1 1.0, and for point-to-point connections between Virtual Components. The PVCI is a subset of Basic VCI as the Basic VCI is a subset of the Advanced VCI.

3.1.2 Organization

This section contains the following chapters:

- Chapter 3.1 provides an overview for the section
- Chapter 3.2 gives a technical introduction to Peripheral VCI
- Chapter 3.3 provides a detailed description of PVCI signals
- Chapter 3.4 defines the PVCI protocol

3.2 Technical Introduction to Peripheral VCI

3.2.1 Initiator – Target Connection

As shown in Figure 3 below, the request contents and the response contents are transferred under control of the protocol, a simple 2-wire handshake.

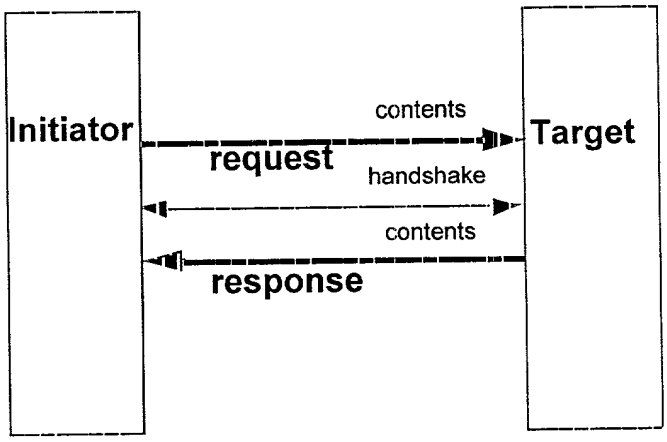

Figure 3. PVCI: The Request and Response Contents are controlled by a Simple Handshake

3.2.2 The Handshake

Note that in Figure 3, the handshake arrow is a double pointed showing that signals in both directions are involved. ('Signal' here is synonymous to 'net', 'wire' or 'port'.) The handshake protocol is aimed at synchronizing two blocks by transferring control information in both directions. The contents-arrow points one way only. The PVCI handshake signals are called **VAL** and **ACK**, as for **Valid** and **Acknowledge**.

Request-contents flow from Initiator to Target, response-contents flow from Target to Initiator. The handshake protocol is introduced below. (See Figures 4 and 5.) Note that the delays are exaggerated in the figures. The triggering events for each transition are expressed with arcing arrows.

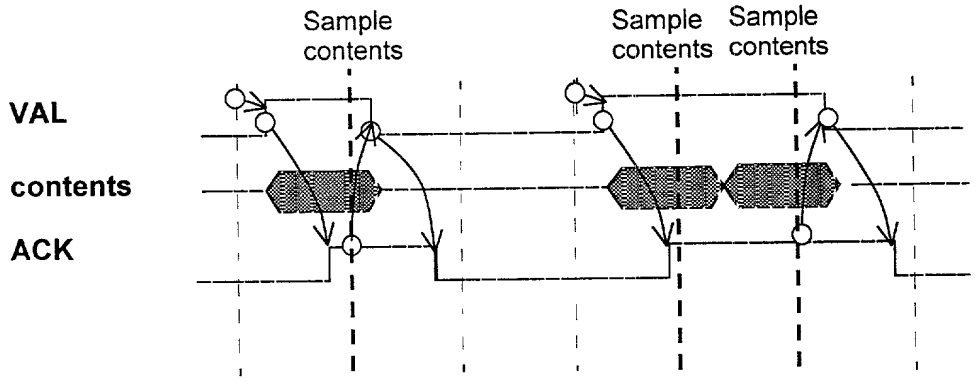

Figure 4. The Control Handshake (Asynchronous ACK)

In Figure 4:

- Vertical dashed lines show rising clock edges.

- **VAL** shows "at the next rising edge, contents can be read". No actual timing is specified here!
- **ACK**, while **VAL** is active at the rising edge of the clock, shows that the contents were read by target, or contents are available to be read from the target. This means the end of the transaction.

VAL and contents must be maintained until **ACK** has become asserted and there is a rising edge of the clock. As indicated in Figure 4 (right hand side), maintaining the **VAL** signal in asserted mode for another clock cycle after **ACK** = 1 means that another request is waiting.

The **ACK** can be either generated asynchronously of the **VAL** as in the Figure 4, or set synchronously at the rising edge of the clock as in the Figure 5. The synchronous **ACK** can be set at the next clock cycle after Valid, or later. (A slow reaction, or late **ACK.**) If asynchronous **ACK** is used, special design considerations are needed to make sure that the **ACK** is stable at the rising clock edge. This means that if the **ACK** is not driven directly by a flip-flop, it must be generated from the **VAL** signal in the target with a minimum amount of logic and wire delay.

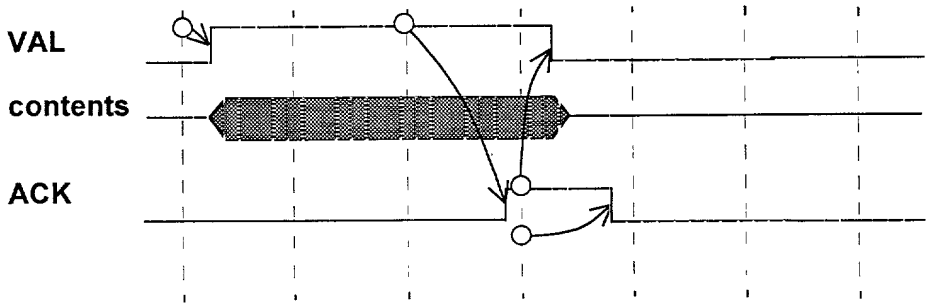

Figure 5. Fully Synchronous Control Handshake: ACK Late by 2 Cycles

The Default Acknowledge

A "default acknowledge" is permitted on top of this protocol. Since the acknowledge-signal is only sampled when **VAL** = 1, it can be asserted (long) before it is needed. Such an early **ACK** has no influence on the protocol behavior. The early **ACK** is merely "don't care" (the signal is not being considered, if **VAL** is not active). The acknowledge-signal can even be tied permanently active, if the target is always able to serve the request in one clock cycle.

3.2.3 Request and Response Contents

Each handshake is used to transfer a *cell* across the interface. *Cell size* is the width of the data passing across a VCI. It will typically be 1,2 or 4 bytes. The cell is synonymous to data word, and the cell size to word length. The cell size is always the same as the size of the particular VC Interface. The Figure 6 shows the signal groups of the PVCI belonging to the request and response. The request contents are the Valid, cell-address, byte-enables within the cell, data to be written, command signal indicating whether we are reading or writing data, and end-of-packet, indicating the end of a burst of several cells (see the description later). The response contents are the acknowledge, response error, and the data read from target. The initiator interface is similar, except that the signal directions are opposite.

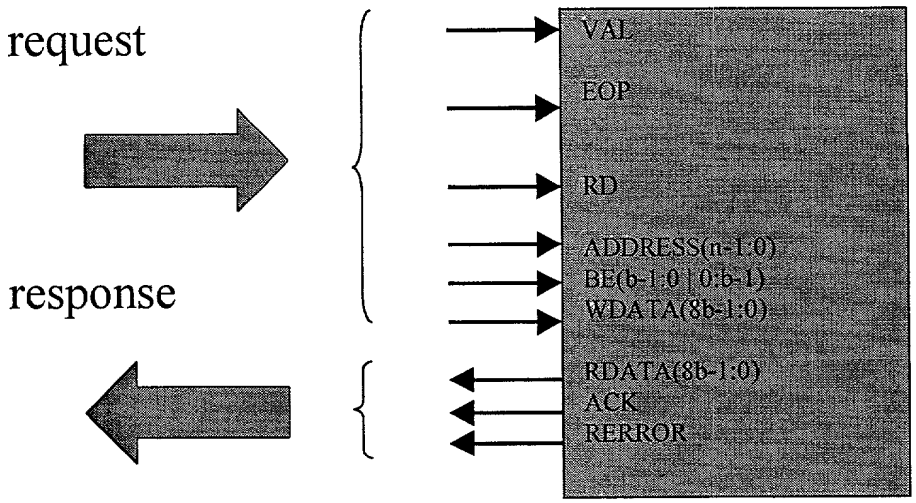

Figure 6. Peripheral VC Interface – Target

3.2.4 Main Peripheral VCI Features:

- Up to 32-bit Address
- Up to 32-bit Read Data
- Up to 32-bit Write Data
- Synchronous
- Allows for 8-bit, 16-bit, and 32-bit devices
- 8-bit, 16-bit, and 32-bit Transfers
- Simple packet, or 'burst' transfer
- Optional Free-BE mode allows transfer of any combination of bytes of a word
- Least Significant Bit is bit "0"
- PVCI has no explicit Endianness
- Natural byte alignment

3.3 Signal Definitions

This chapter defines the PVCI signals. First, the signals are summarized in table format. A textual definition of each signal follows. All signals included in Table 1 below are assumed to be active-high signals unless explicitly indicated otherwise. It is recommended that all signal outputs are stable before Early. It can be assumed that all inputs are stable before Late.

System Signals

Signal Name	Driver	Receiver	Width	Comments
CLOCK	System	Initiator/Target	1	Supplied by the system. Interface is synchronous to the rising edge only.
RESETN	System	Initiator/Target	1	Supplied by the system. Active low reset. De-asserts VAL and ACK.

Handshake, Flow Control, and Shaping

Signal Name	Driver	Receiver	Width	Comments
VAL	Initiator	Target	1	This pair of signals provides flow control for a cell transferring across the VC Interface, and validates all signals associated with that cell. VAL==1: indicates that the initiator has a cell available. ACK==1: indicates that the target can complete the operation on the cell. The cell is therefore transferred when VAL==ACK==1
ACK	Target	Initiator	1	
EOP	Initiator	Target	1	EOP==1 indicates that the current cell is the last one in a series of cells accessed at contiguous addresses. Can be interpreted as an inverted burst signal.

Operation Information

Signal Name	Driver	Receiver	Width	Comments
ADDRESS [N-1:0]	Initiator	Target	N	N is a parameter based upon the capabilities of the target and is defined and fixed at the time of component instantiation. The most significant bit of the address is carried by bit W-1, and the least significant bit is carried by bit 0. See 3.3.3 for use of low-order address bits.
RD	Initiator	Target	1	This is a single bit code giving the operation type: READ==1: Read data from the target peripheral READ==0: Write data to the target peripheral
BE [b-1:0 |0:b-1]	Initiator	Target	b	This is a field with b-bits, one for each byte, which indicates which bytes of the word being transferred are enabled. The usage restrictions of BE are listed in 3.4.3.

WDATA [8b-1:0]	Initiator	Target	8b	This is the data, which is transferred with write operations to the target. For the Peripheral Interface, the allowed values of b are 4, 2, 1. Bit 8b-1 is the most significant bit, and bit 0 is the least significant bit. Byte [8b-1:8b-8] represents the most significant byte. For VCs supporting a data size which is not a power of two, the next larger supported b will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide PVCI with the 4 Most Significant bits tied to logic zero.
RDATA [8b-1:0]	Target	Initiator	8b	This is the data, which is returned from the target with read operations. Since the peripheral interface has no pipelining, RDATA is validated by the target when it asserts ACK. It is defined in the same way as WDATA above.
RERROR [E:0]	Target	Initiator	E+1	Indicates error in transfer. Optional error extension bits indicate nature of the error. E is defined by parameter ERRLEN. It is zero or more

Table 1: Peripheral VCI Signals

3.3.1 System Level Signals

Clock

Signal Name:	Clock
Signal Abbreviation:	CLOCK
Polarity:	Active at Positive edge
Driven By:	System
Received By:	VCI Initiator, VCI Target

This signal provides the timing for the Virtual Component Interface and is an input to both the initiator and target that are connected via the PVCI. All initiator and target output signals, unless otherwise specified, are asserted/de-asserted relative to the rising edge of CLOCK and all initiator and target inputs are sampled relative to this edge.

Reset

Signal Name:	Reset
Signal Abbreviation:	RESETN
Polarity:	Asserted Negative
Driven By:	System
Received By:	VCI Initiator. VCI Target
Timing:	Asserted > 8 clock cycles

This signal is used during power-on reset and is used to bring the PVCI to an idle or quiescent state. This idle state is defined as the PVCI state in which:

1. The VAL signal is de-asserted
2. The ACK signal is de-asserted

The system must guarantee that RESETN is asserted for at least eight cycles of CLOCK (More if RESETLEN parameter is set.).

3.3.2 Control Signals

Valid

Signal Name:	Valid
Signal Abbreviation:	VAL
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until ACK == 1 and next rising edge of CLOCK.

The VAL signal is driven by a VCI initiator to indicate that there is a valid address, data, and command on the PVCI. All of the initiator control signals are qualified by VAL. The initiator keeps VAL asserted, and all of its control signals valid and stable, until it receives the ACK signal from the target. The initiator should not assert VAL unless the current transaction is intended for the target. Thus, the initiator may need to perform address decoding on its on-chip bus side to generate VAL for the target, thereby accomplishing device selection.

Acknowledge

Signal Name:	Acknowledge
Signal Abbreviation:	ACK
Polarity:	Asserted Positive
Driven By:	VCI Target
Received By:	VCI Initiator
Timing:	Asserted and negated at rising edge of CLOCK. The signal must change before Late.

The ACK signal is asserted by the target to indicate the completion of a transfer between the initiator and target. In the case of write operations, this means that the target has accepted the data which is on the write data bus or will do so at he end of the current clock cycle. In the case of read operations, the assertion of the ACK by the target indicates that the target has placed data to be transferred to the initiator on the read data bus. The transaction completes as soon as the rising edge of CLOCK samples ACK. The target may de-assert the ACK by the next rising edge of CLOCK unless a new command has been initiated by the initiator, or default acknowledge is used. Similarly, the initiator de-asserts VAL by the next rising edge of CLOCK unless it is presenting a new command.

Read / Not Write

Signal Name:	Read / Not Write
Signal Abbreviation:	RD
Polarity:	Read at positive, write at zero
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until ACK == 1 at rising edge of CLOCK .

This signal is a one-bit command asserted by the initiator and indicates the direction of the requested transfer. RD indicates that the requested transfer is a *read* if it is asserted high and a *write* if it is asserted

low. This signal must be valid any time that the VAL signal is asserted. A read transfer is a request for the target to supply data on RDATA to be read into the initiator. A write transfer is a request for the target to accept data on WDATA from the initiator.

End-of-Packet

Signal Name:	End-of-Packet
Signal Abbreviation:	EOP
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until ACK = 1 at rising edge of CLOCK

The EOP signal is de-asserted by the nitiator to indicate that the transfer being performed will be followed with a transfer by the initiator to the next higher cell address. This signal is used by the target device to preset address in order to improve the data transfer performance. Thus this signal can be interpreted as an inverted burst signal. The burst is similar
to a packet in the Basic VCI, except that rather than prescribing a strict atomicity, it merely indicates a contiguous address behavior. The burst transfer is completed once a cell is transferred with the EOP signal asserted, or when the Target signals an Abort error. See the definition of ADDRESS signal for the legal address behavior during a burst.

Basic VCI packet with CONTIG signal asserted high and WRAP signal asserted low can be mapped to a PVCI burst.

3.3.3 Address and Data Signals

Address

Signal Name:	Address
Signal Abbreviation:	ADDRESS[n-1:0]
Polarity:	N/A
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until ACK == 1 at rising edge of CLOCK

The PVCI initiator uses ADDRESS to identify which target-resource the current transaction acts upon. The N-lines of address bus form a binary number that represents an address. N is a parameter based upon the capabilities of the target and is defined and fixed at the time of component instantiation. The most significant bit of the address bus will be carried by bit N-1 and the least significant bit of the address will be carried by bit 0. ADDRESS[n-1:m] provides a cell address where the cell size is specified by the PVCI's total data width. M equals to 0 for cell size 1, 1 for cell size 2, and 2 for cell size 4. Sub-cell addressing is handled by the byte enable signals.

It is permissible to supply the low-order address bits to allow the original full intention of the transfer to be maintained. This will rely on knowledge of the endianness of the initiator. This additional information is NOT required to complete the operation, which is completely defined by a cell-aligned address and byte enables (i.e. the low-order address bits may be tied to logical zero). However, in some systems, some efficiency gains may be possible by taking advantage of this information.

When the EOP signal is low, the ADDRESS of the next cell must be cell_size + ADDRESS of the current cell, and the ADDRESS of the current cell must be aligned with the cell boundary.

Byte Enable

Signal Name:	Byte Enable
Signal Abbreviation:	BE[**b**-1:0|0:**b**-1]
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until ACK == 1 at rising edge of CLOCK

BE is a **b**-bit field that indicates which bytes of the cell being transferred are enabled. The **b** equals to total data width of the PVCI / 8. These signals must be valid any time that the VAL signal is asserted. The usage of byte enables is described in the Peripheral VCI as described in Section 3.4.3 *Data Formatting and Alignment.*

BE[3:0] is used for little endian VCs where the LSB (Least Significant Byte) is labeled Address 0.
BE[0:3] is used for big endian VCs where the MSB (Most Significant Byte) is labeled Address 0.

Write Data

Signal Name:	Write Data
Signal Abbreviation:	WDATA[8**b**-1 : 0]
Polarity:	N/A
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until ACK == 1 at rising edge of CLOCK

The write data lines are driven by the VCI initiator and are used to transfer write data from an initiator to a target device. Write data consists of **b** logical byte lanes, based upon the capabilities of the target and is defined and fixed at the time of component instantiation. Allowed vales of **b** are 8, 16, and 32, allowing 1, 2 or 4 byte lanes. Bit 8***b**-1 is the most significant bit of the most significant byte and bit 0 is the least significant bit of the least significant byte. The write data lines must contain valid write data while the VAL signal is asserted and the READ is indicating a write transfer.

For VCs supporting a data size that is not an eight bit increment, the next larger supported bus size will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide PVCI with the 4 Most Significant bits tied to logic zero.

Read Data

Signal Name:	Read Data
Signal Abbreviation:	RDATA[8b-1: 0]
Polarity:	N/A
Driven By:	VCI Target
Received By:	VCI Initiator
Timing:	Asserted at rising edge of CLOCK until ACK == 1 at rising edge of CLOCK

The read data lines are driven by the VCI initiator and are used to transfer read data from a target to an initiator device. Read data consists of **b** logical byte lanes, based upon the capabilities of the target and is defined and fixed at the time of component instantiation. Allowed vales of **b** are 8, 16, and 32, allowing 1, 2 or 4 byte lanes. Bit 8***b**-1 is the most significant bit of the most significant byte and bit 0 is the least significant bit of the least significant byte. The read data lines must contain valid read data while the ACK signal is asserted and the RD is indicating a read transfer.

For VCs supporting a data size that is not an eight bit increment, the next larger supported bus size will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide PVCI with the 4 Most Significant bits tied to logic zero.

3.3.4 Error Signals

Response Error

Signal Name:	Response Error
Signal Abbreviation:	RERROR[E:0]
Polarity:	Positive
Driven By:	VCI Target
Received By:	VCI Initiator
Timing:	Asserted at rising edge of CLOCK when ACK == 1 at rising edge of CLOCK.

Error signal is valid only when ACK=1, with the following meaning:
For ERRLEN = 0 (E=0)
RERROR=0: Normal (no error)
RERROR=1: General data error. The entire packet is considered bad.

For ERRLEN = 1 (E=1)
RERROR = 00: Normal (no error)
RERROR = 01: General data error. The entire packet is considered bad
RERROR = 10: Reserved
RERROR = 11: Abort Disconnect

For ERRLEN = 2 (E=2)
RERROR =000: Normal (no error)
RERROR = xx0: Reserved
RERROR = 001: General data error. The entire packet is considered bad
RERROR = 011: Reserved
RERROR = 101: Bad data (retry)
RERROR = 111: Abort Disconnect

After receiving an error, the Initiator may or may not continue with the current packet. If it chooses to end the packet prematurely, it can do so by asserting EOP. After this, it can choose not to try a transfer anymore (Abort) or Retry part or all of the transfer. For any error, the Target must process the subsequent pending cells and packets with the normal protocol, i.e. it must continue sending responses until it has processed the EOP with or without further errors signaled. In general, the RERROR is more informative than prescriptive, and the Target may not assume any special behavior from the Initiator. The Initiator is anyhow encouraged to act responsibly, when it meets an error.

While sending a non-Abort error, the target should continue with the burst, with or without further errors until it receives a cell with EOP. The Initiator can choose to continue with the burst or set the EOP on the next cell. In the case of Bad data, just the data of error may be considered bad. If ERRLEN=2 is supported, the Target must be capable of accepting a retried request to the same address where it sets Retry-error, but the Initiator can choose to retry a cell, burst or nothing at all. It is strongly recommended that all PVCI components would support at least one bit error. The PVCI target is strongly recommended to signal an error when it receives a request it does not support.

3.4 PVCI Protocol

3.4.1 Operation Types

Transfer Request

The following is a list of default transaction types that may be initiated by an initiator across the Peripheral Virtual Component Interface (PVCI). An initiator should not initiate an operation of a width that the target does not support.

- Read8: Read one byte. The byte may be on any byte lane expressed with the BE field. Supported by all PVCI components

- Read16: Read two bytes. The bytes must be on contiguous byte lanes, and are expressed with the BE field. The operations must be aligned to 16-bit sub-word boundaries, i.e. transfer with BE==0110 is not allowed. Supported by 2- and 4-byte PVCI components.

- Read32: Read four bytes. Supported by 4-byte PVCI components.

- Read N Cells: Read a packet of N cells. The addresses of consecutive cells must be ascending, cell aligned, and consecutive. The byte enables of individual cells may have any legal values. The last cell is indicated with the EOP signal. Each cell of a packet is individually requested and handshaken. Only one pending request for a cell is allowed at a time.

- Write8: Write-operations are similar to read-operations, except for the data direction

- Write16

- Write32

- Write N Cells

All transfers are packet transfers of 1 cell or more. A packet transfer of length 1 is indistinguishable from a single transfer. Neither the initiator nor the target need support packets that cross address selection boundaries.

The optional Free-BE mode operations are as follows:

-Read: Read any combination of the bytes in the cell as expressed by BE field.

-Write: Write any combination of the bytes in the cell as expressed by BE field.

Transfer Response

The following is a list of responses that are allowable by a target across the Peripheral Virtual Component Interface (PVCI).

- Not Ready
- Transfer Acknowledged
- Error

3.4.2 Handshake Protocol

The two-wire handshake was introduced in the chapter 3.2.2. The Peripheral VCI handshake protocol is a subset of the Basic VCI handshake protocol (See 4.2.2). Connecting a Basic VCI target to a Peripheral VCI initiator is trivial; the extra signals are tied into constants. The performance achieved is similar to native Peripheral connection. It is also possible to connect Basic VCI initiator to a Peripheral VCI target, if the initiator knows the limitations of the target.

The VAL-ACK pair of signals provides a flow control for a cell transferring across the VC Interface, and validates all signals associated with that cell. See waveform pictures below for examples.

- VAL==1: Indicates that the initiator is presenting a request.
- ACK==1: Indicates that the target is ready to present the response.

Rules :

R1 : **not changeable** : Once an initiator has asserted VAL, it is not permissible to de-assert it or to change any request field, until acknowledged.

R2 : **default_ack** : ACK is permitted to be asserted when VAL is low.

R3 : **valid response** : The target must be presenting stable response fields at the rising edge of the clock while both ACK and VAL are asserted.

The initiator must keep VAL asserted, and all of its control signals valid and stable, until it receives the ACK signal from the target. The initiator should not assert VAL unless the current transaction is intended for the target.

Packet Transfer

The packet (burst) transfer is meant for making transfer of block of cells with consecutive addresses more efficient. While the EOP signal is de-asserted during a request, the address of next request will be ADDRESS+cell_size. The ADDRESS must be aligned to the cell boundary. The packet transfer is completed once a cell is transferred with the EOP signal asserted, or when the Target asserts RERROR = Abort. In terms of the Basic VCI operations, the Peripheral VCI burst equals to a packet transfer with CONTIG signal asserted and WRAP signal de-asserted (See 4.3). The byte enable signals may have any legal combination in each cell of the packet.

Examples:

The following picture shows waveforms of read and write operations to a target, which needs two clock cycles to complete the read, and one cycle to complete the write. If the RERROR signal is not drawn, it is 0 ('Normal') for all the examples.

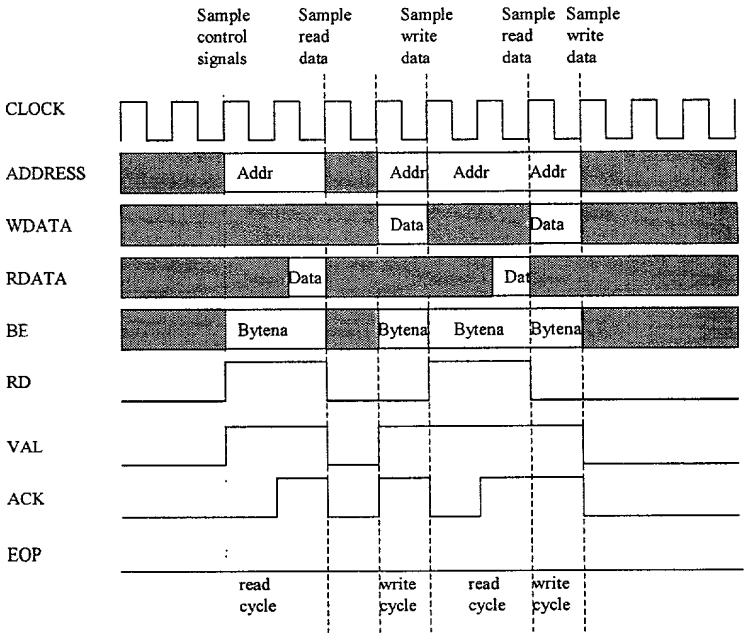

Figure 7. PVCI Read and Write

The following picture shows waveforms of read and write operations to a target, which has default acknowledge, i.e. requires single cycle to complete read and write operation.

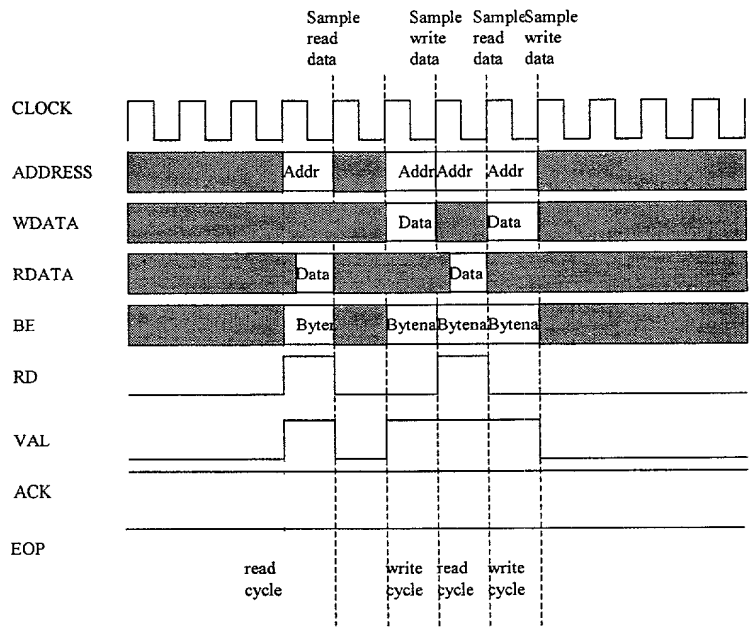

Figure 8. PVCI Read and Write with Default Acknowledge

The following picture shows how the EOP signal can be used to indicate address predictability to a similar target, which has an internal address counter to support pre-fetching read data. Notice that the cell size is 4 in the example, so the ADDRESS is incremented by 4 at each read. In this example, the target can respond to the read in two clock cycles in single transfer, and in one cycle in burst transfer. The ACK signal can be generated of the VAL, RD, and EOP signals.

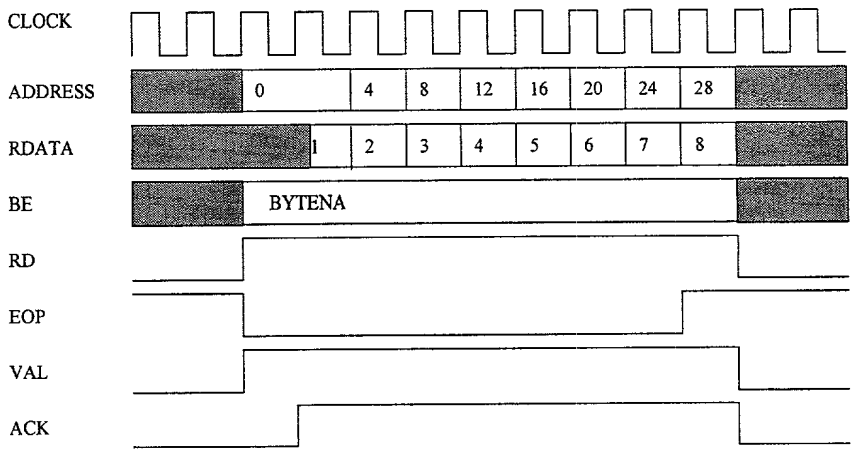

Figure 9. PVCI Burst Read

The following picture illustrates how 'Abort' error response will terminate a burst transfer.

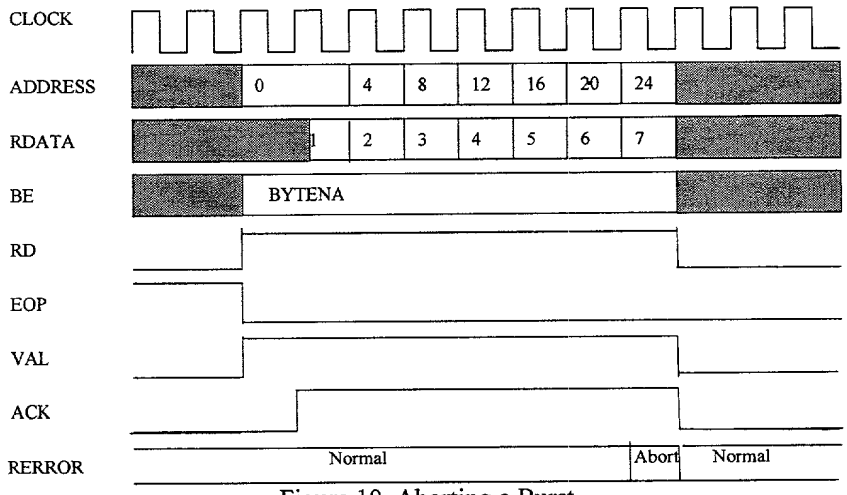

Figure 10. Aborting a Burst

3.4.3 Data Formatting and Alignment

This section describes the PVCI conventions used for bit / byte ordering and alignment. The Peripheral VCI defines bit "0" as the least significant bit of a vectored field such as WDATA, RDATA, or ADDRESS, bit "8b-1" as the most significant data bit of each data bus, and bit "N-1" as the most significant address bit. N is defined to be the number of physical ports or wires associated with a particular instantiation of a PVCI.

The PVCI is endian independent at the interface. Even though the naming convention may imply a little endian interface, the DATA[7:0] means address == 0 for little endian, and address == 3 for big endian (in case of a 32-bit interface). PVCI component on the other hand must choose an endianness it wants, and declare this to the system (unless the component is a memory, where it does not matter). When any component interprets a byte address it makes a decision on which byte lane to use.

The Peripheral VCI will use natural alignment to support peripherals and VC's that handle multiple size transfers (e.g. byte as well as word transfers). This means that transfer sizes that are smaller than the physical width of the data lines (WDATA or RDATA) will occur in their natural byte lanes and not be right or left justified. The byte enable lines (BE) indicate, which byte lane(s) contain the desired data. Table 2 shows the various data alignments defined for the peripheral VCI data transfers. The entries labeled "xx" are don't cares, which provide flexibility and support for byte replication if desired.

The PVCI standard has two operational modes related to byte enables:

Byte Enables in Default Mode

Every PVCI component must support usage of the byte enable lines that are restricted to the contiguous cases. Referring to the following table and the 32-bit example, the allowable patterns for the byte enables are 0000, 0001, 0010, 0100, 1000, 0011, 1100, and 1111. Patterns such as 1011 or 1101 are NOT allowed.

VCData bus size	Xfer size	BE [3:0|0:3]	Data[31:24]	Data[23:16]	Data[15:8]	Data[7:0]
32	32	Others	Undefined			
		0000	XX	XX	XX	XX
		1111	Byte	Byte	Byte	Byte
	16	0011	XX	XX	Byte	Byte
		1100	Byte	Byte	XX	XX
	8	0001	XX	XX	XX	Byte
		0010	XX	XX	Byte	XX
		0100	XX	Byte	XX	XX
		1000	Byte	XX	XX	XX
VC Data bus size	**Xfer size**	**BE [1:0|0:1]**	**Data[31:24]**	**Data[23:16]**	**Data[15:8]**	**Data[7:0]**
16	16	00			XX	XX
		11			Byte	Byte
	8	01			XX	Byte
		10			Byte	XX
VC Data bus size	**Xfer size**	**BE[0]**	**Data[31:24]**	**Data[23:16]**	**Data[15:8]**	**Data[7:0]**
8	8	0				XX
		1				Byte

Table 2: Peripheral VCI Data Alignment

Both in big or little endian mode, the BE[0] always corresponds to the byte address 0 in the cell.

Byte Enables in Free-BE Mode

The Free-BE mode is an optional operation mode for PVCI components. It is legal in this case to support any BE pattern. This mode can be used to make connections to some buses more efficient (e.g. a bus that support 3-byte wide transfers). A Target, which supports the Free-BE mode, automatically supports the Default mode. A PVCI Initiator must not require that the Target supports Free-BE mode, but it can take an advantage of it, if it does. The operation mode is selected in component instantiation; it is not a run-time parameter.

See VCI Parameters in the Design Guidelines section of the VCI Standard.

4. Basic VCI

4.1 Overview

4.1.1 Scope

This section defines a Basic Virtual Component Interface (BVCI) to be used in conjunction with on-chip system buses, and for point-to-point connections between Virtual Components. The BVCI is a subset of Advanced VCI. The BVCI is designed to fulfill most on-chip interfacing needs protocol wise.

4.1.2 Organization

This section contains the following chapters:

- Chapter 4.1 provides an overview for the section
- Chapter 4.2 gives a technical introduction to Basic VCI
- Chapter 4.3 provides a detailed description of BVCI signals
- Chapter 4.4 defines the BVCI protocol in great detail.

4.2 Technical Introduction to the Basic VCI

4.2.1 Initiator – Target Connection

As shown in Figure 11 below, the request contents and the response contents are separately transferred under control of the protocol, a simple handshake, as introduced in Section 4.2.2.

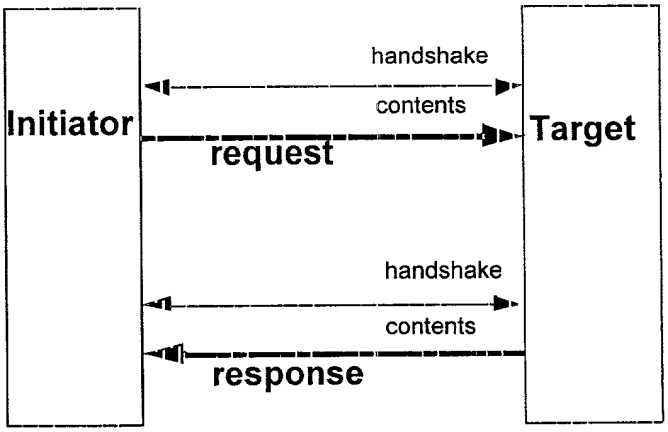

Figure 11. VCI: The Request and Response Each Consist of a Handshake and Contents

4.2.2 The Handshake

The BVCI handshake is different from the PVCI one: The request and response handshakes are completely independent of each other. The handshake protocol is aimed at synchronizing two blocks by transferring control information in both directions. In the request, the handshake signals are called **CMDVAL** and **CMDACK** (**RSPVAL** and **RSPACK** in the response). Those are acronyms for Command Valid, Command Acknowledge, Response Valid, and Response Acknowledge, respectively.

Request-contents flow from Initiator to Target; response-contents flow from Target to Initiator. The handshake protocol is introduced below. (See Figure 12.) More precise details are provided later in the text. This introductory section only serves to provide a basic understanding of the overall process.

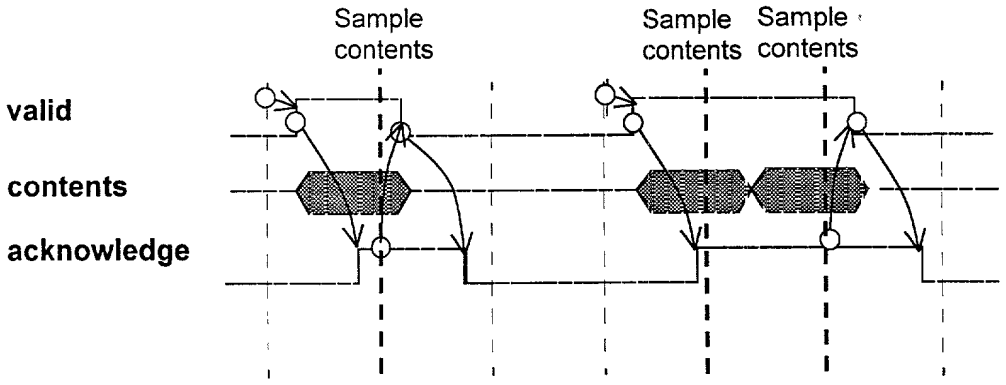

Figure 12. The Control Handshake for Both Request and Response (Asynchronous ACK)

In Figure 12:

- Vertical dashed lines show rising clock edges.
- **VAL** shows "at the next rising edge, contents can be read". No actual timing is specified here!
- **ACK**, where there is a coinciding **VAL**, shows "contents have been read". No actual timing is specified here!

As indicated in the figure (right hand side), maintaining the **VAL** signal in asserted state for another clock cycle after **ACK** = 1 means that another request or response is ready for reading. **VAL** and contents must be maintained until the next rising clock edge after the **ACK** has become asserted.

The **ACK** can be either generated asynchronously of the **VAL** as in the Figure 12, or set synchronously at the rising edge of the clock as in the Figure 13. (A slow reaction, or late **ACK.**) If asynchronous **ACK** is used, special design considerations are needed to make sure that the **ACK** is stable at the rising clock edge. For this reason, it is not recommended to use an acknowledge, which is not generated synchronously except with "default acknowledge" behavior (See below).

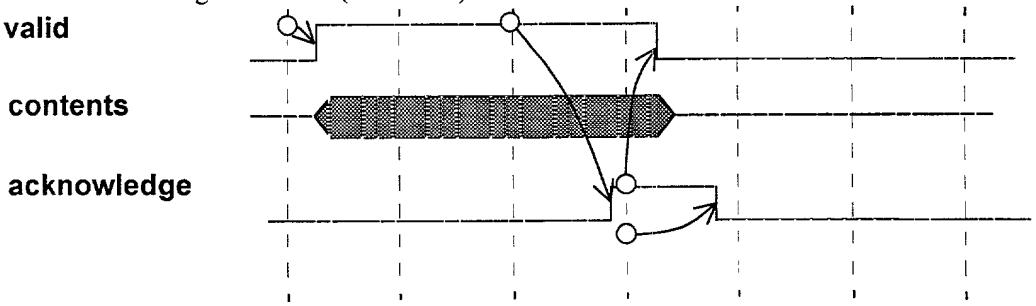

Figure 13. Fully Synchronous Control Handshake: ACK Late by 2 Cycles

4.2.3 The Default Acknowledge

A "default acknowledge" is permitted on top of the protocol. Since the Acknowledge signal is only sampled when **VAL** = 1, the Acknowledge signal can be asserted (long) before it is needed. Such an early **ACK** has no influence on the protocol behavior. The early **ACK** is merely "don't care" (the signal is not

being considered unless **VAL** is active at a clock edge). This makes it possible to tie **ACK** permanently active.

4.2.4 Cells, Packets, and Packet Chains

4.2.4.1 Cell

Each handshake is used to transfer a cell across the interface. The cell size is the width of the data passing across a VCI. It will typically be 1, 2, 4, 8 or 16 bytes.

4.2.4.2 Packet

Cell transfers can be combined into packets, which may map onto a burst on a bus. A VCI operation consists of a request packet and a response packet. As noted in Section 2, although the responses in a packet arrive in the same order as their requests, there is no further relation between the timing of the series of requests and the series of responses. The protocol is a split protocol. Packets have been introduced for three reasons:

1. When connecting the BVCI to a bus: If the underlying bus system is aware that more operations are to follow, then no bus arbitration is required between operations of one packet. This may well gain valuable bus capacity and speed.

2. Packets are atomic. That is, the point-to-point connection is maintained over the packet, no matter what is in between Initiator and Target - a complex interconnection set-up consisting of one or more buses or "nothing at all". This set-up, for example, allows for the monopolizing of a buffer.

   Note, though, that long packets block any other use of the connection system. **It is strongly advised to keep packets as short as possible to prevent degradation of system performance.**

3. The "Advanced VCI", not yet defined, needs a multi-operation building block for more sophisticated packet chains. The packet thus is needed for upward compatibility.

Packets are similar in concept to "frames" in PCI, where a connection is established to enable data to be transferred until the Initiator indicates the termination by closing the frame.

4.2.4.3 Packet Chain

Packets can be combined into chains, to allow longer chains of operations to go uninterrupted, as long as no higher priority operation claims part of the connection, such as a bus. Packet chains thus produce the same bus efficiency that packets provide, without actually excluding any other usage of the connection system. The VCI Initiator may merely "notice" that the **CMDACK** to the first operation of a new packet is withheld for a long time. After this pause, the accepting of requests proceeds as if no other usage of the connection system had been served. To obtain this behavior, **CMDVAL** should be held asserted between the packets of a chain. There is no upper limit to the length of a packet chain.

4.2.5 Request Contents

Request contents are partitioned into three signal groups:

1. Opcode to specify the nature of the request (roughly: read or write),

2. Packet Length and Chaining to control packets, and

3. Address and Data to further detail a read or write action.

Request contents are validated by the **CMDVAL** signal. An introductory description of request contents is provided below. A more detailed description follows.

4.2.5.1 Operation Code (opcode)

This subset of the request contents is constant for all the operations in a packet.

- **Command**: The two bit field "CMD" can specify no-operation, read operation, write operation, or read locked.

- **Flags** (address algorithm indicators): When none of the flags is asserted, there is no predefined relationship among the addresses of subsequent operations in a packet or in a packet chain. Three flags, though, can indicate a predefined algorithm among subsequent addresses. This allows Targets to compute addresses before they actually arrive via the bus and thus, in some cases, to gain valuable clock tics. The address, even when obeying a predefined algorithm, is sent with each operation, allowing cost effective targets to function without address prediction. The three flags are:

  1. **Contiguous**: Indicates that the addresses within a packet increasing in a contiguous manner.

  2. **Wrap** (only valid with contiguous addressing): The increase is done modulo the packet length, and only when packet length is power of two. This allows for a cache line refill starting with the missing word rather than with the word at the lowest address.

  3. **Constant** (no change in address): This mode is useful for the case when a series of FIFOs have contiguous addresses and a packet or a chain is used to empty or fill just one of these FIFOs.

(In the Advanced VCI, more such flags may be defined.)

4.2.5.2 Packet Length and Chaining

This subset of the request contents is constant for all the operations in a packet, apart from the indicator "End of Packet", which is only asserted in the last cell of a packet. Contents include:

- **Packet Length**: The length of the packet expressed in bytes.

- **End of Packet**: This bit is asserted in the last operation of a packet.

  - The bit could be considered redundant in presence of a defined packet length, but it allows a Target to be designed without its own explicit remaining length counter. Furthermore, end-of-packet is a necessity with an undefined packet length.

- **Chain Length**: Presents the amount of packets yet to come in a packet chain.

  - Chain length = 0 thus represents a one-packet chain. There is no need for an "eoc" (end of chain).

- **Chain Fixed:** Indicates that the opcode fields (see Section 4.2.5.) and packet length are equal for all packets of the chain. Otherwise each packet of a chain has its own such fields and flags.

4.2.5.3 Address And Data

This subset of the request contents is issued anew for each cell within a packet. Contents include:

- **Address:** The address field designates the Target if several Targets can be reached through the VCI, and the detailed location within the target to which the request is made.

  - This standard does not prescribe how Target addresses are allocated to address-space. For example, four FIFOs may be assigned consecutive word addresses while memories may occupy Megabytes.

  - The address is updated for every operation in a packet even if the address algorithm is specified by means of the flags.

- The address is specified as the lowest byte address of the concerned data.

- **Byte enable**: The main role of byte enable is in write operations. Each asserted bit in this field designates a byte in the cell to be actually written into the target. Each non-asserted bit marks a byte are not to be overwritten.

  In read operations, byte enable is needed in those wrappers or bridges where cell length or data alignment changes.

  There are two separate naming conventions for byte enable, corresponding to a natural endianness.

  - **BE**[3:0] is used for little endian VCs where the LSB (Least Significant Byte) is labeled Address 0.

  - **BE**[0:3] is used for big endian VCs where the MSB (Most Significant Byte) is labeled Address 0.

- **Write data**: The write data field is only used in write operations. The data lines carry the bytes to be copied to the target.

  - Data are "naturally aligned": the byte corresponding with byte address modulo cell size = 0 is at byte line 0 on the bus.

4.2.6 Response Contents

Each request has its response. The response contents are validated with the **RSPVAL** signal. Contents include:

- **Response Error**: The Response Error field indicates whether the request could be handled correctly.

- **Read Data**: The data returned as a result of a read request. This field has no meaning in write operations.

  - Data are "naturally aligned". The byte corresponding with byte address modulo cell size = 0 is at byte line 0 on the bus.

4.3 Basic VCI Signal Descriptions

This is the first section of detailed BVCI technical description. Descriptions of the signals used between the Initiator and Target over the VCI are provided in the following.

4.3.1 Signal Type Definition

Table 3 specifies the signal types that are used in Section 4.3. They are defined from the point of view of the devices rather than the wrapper or arbiter.

Type	Description
IA	Input to all devices
IT	Generated by the Initiator and sampled by the Target
TI	Generated by the Target and sampled by the Initiator
MA	Mandatory signal for both Initiator and Target
MI	Mandatory signal for the Initiator but an optional signal for the Target
MC	Mandatory signal for supporting chaining function for both Initiator and Target

Table 3: Signal Type Definition

Signals that do not have one of the mandatory descriptors listed above are optional and do not support the indicated function.

4.3.2 Signal Parameters

Table 4 specifies parameters that are used in Section 4.3.

Parameter	Description
B	Number of bytes in a cell (must be a power of 2)
K	Number of bits in the **PLEN** field (maximum value is 9)
N	Number of bits in the **ADDR** field
E	Number of bits in the RERROR field
Q	Number of bits in the **CLEN** field

Table 4: Signal Parameters

4.3.3 Signal Directions

The Figure 14 diagrams the signal directions between the Initiator and the Target.

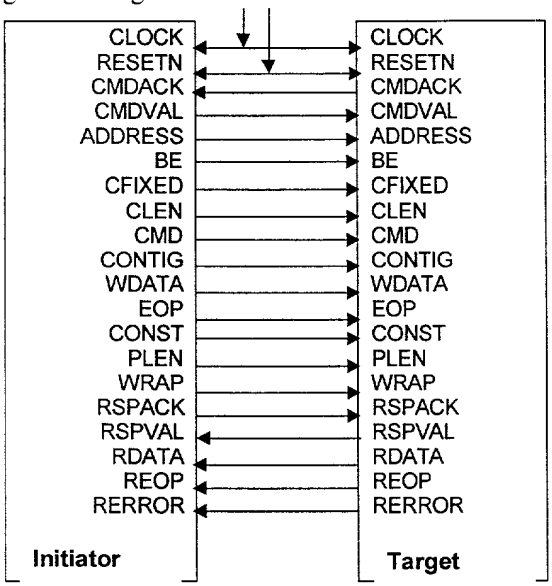

Figure 14. Diagram of VCI Signal Directions

4.3.4 Signal List

All signals included in Table 5 below are assumed to be active-high signals unless explicitly indicated otherwise. It is recommended that all signal outputs are stable before Early. It can be assumed that all inputs are stable before Late. A detailed signal description follows the summary table.

Table 5: Signal List

Name	Type	Description
		Global Signals
CLOCK Clock	IA MA	**CLOCK** provides the timing for all transactions. All signals are sampled on the rising **CLOCK** edge. All timing parameters are initiated with respect to this edge.
RESETN Reset	IA MA	Reset is used to bring all devices up on a common signal. **RESETN** is an active low signal and must be asserted for a minimum of 8 **CLOCK** cycles. The rising edge of **RESETN** is synchronous to the rising edge of **CLOCK**.
		Request Handshake
CMDACK Command Acknowledge	TI MI	Acknowledge is used by the Target to indicate to the Initiator that a given cell can be transferred. Hence, a cell is transferred from the Initiator when both **CMDVAL** and **CMDACK** are asserted.
CMDVAL Command Valid	IT MI	Valid indicates that the Initiator wishes to perform a cell transfer to the Target. The cell is transferred when both the **CMDVAL** and **CMDACK** signals are asserted.
		Request Content
ADDRESS[n-1:0] Address	IT MA	**ADDRESS** is the address of the request generated by the Initiator and received by the Target. The address updates for every cell transferred within a packet and must remain within the address space of a single Target. The pattern of addresses that are permissible are defined by the flags (**CONTIG**, **WRAP**, and **CONST** signals). **ADDRESS** contains the lowest byte address for the first transfer in the packet. For all cells after the first transfer, **ADDRESS** is aligned to a cell boundary. The combination of a cell-aligned address and byte enables is sufficient to perform the transfer correctly. However, the addition of extra information in the first address may allow performance advantage in some systems. Note that a non cell-aligned address is endian dependent.
BE [b-1:0|0:b-1] Byte Enable	IT MA	Byte Enable indicates which bytes of the cell being transferred or requested by the initiator are enabled.
CFIXED Chain Fixed	IT MC	Chain Fixed indicates that the opcode (**CMD**, **CONTIG**, **WRAP**, and **CONST**) and **PLEN** fields will be constant across the chain and the address field behavior will be the same among packets within a chain.
CLEN[q-1:0] Chain Length	IT MC	Chain Length indicates the number of packets remaining in a chain. The last packet transferred in a chain will have a zero **CLEN** value. The **CLEN** value can also be tied off to zero if packet chaining is not required.
CMD[1:0] Command	IT MA	Command is a 2-bit code defining the operation type being attempted by the Initiator to the Target and is encoded as follows: • 00b = NOP, no data is actually transferred (optional) • 01b = READ, data is requested by the Initiator from the Target • 10b = WRITE, data is transferred from the Initiator to the Target • 11b = LOCKED READ, data is requested by the Initiator from the Target with the added function of locking out access to at least the particular cell until a WRITE packet is transferred to the same cell (optional)
CONTIG Contiguous	IT	**CONTIG** indicates that the sequence of addresses that will be performed within the packet is contiguous. When CONTIG is asserted, the address is normally increased by the cell size in bytes. If a packet does not start or end at a cell boundary, the first or last transfers contain less bytes than a cell. In such cases, the address is increased not by the length of a cell but by the amount of bytes actually transferred in such a first or last cell.

Name	Type	Description
WDATA[8b-1:0] Data	IT MA	**WDATA** is the data transferred by the Initiator to the Target during a WRITE command. The actual data bits that are enabled during a given cell's transfer are defined by the byte enables. The most significant bit is always on the LHS – bit 8b-1. The least significant bit is always on the RHS – bit 0.
EOP End Of Packet	IT MI	End Of Packet is asserted on the last cell of a packet indicating that all cells associated with the given packet have been transferred.
CONST Constant	IT	**CONST** indicates that the address will remain constant throughout the entire packet. When **CONST** is asserted, **CONTIG** and **WRAP** are ignored.
PLEN[k-1:0] Packet Length	IT	Packet Length indicates the length of the packet in bytes. The valid range for **PLEN** is 1 to 2^k-1 (1 to 511 given that k is limited to 9). A value of 0 for **PLEN** can be used to indicate that the packet length is undefined (no implied packet length).
WRAP Wrap	IT	**WRAP** is used in conjunction with **CONTIG** to indicate how addresses that increment past the boundary indicated by **PLEN** are handled. If **WRAP** is asserted and **PLEN** has only a single bit set (indicating a power of 2 packet size), the address will wrap around.
Response Handshake		
RSPACK Response Acknowledge	IT MA	Response Acknowledge is used by the Initiator to indicate to the Target that a given cell will be transferred. Hence, a response cell is transferred from the target when both **RSPVAL** and **RSPACK** are asserted.
RSPVAL Response Valid	TI MA	Response valid indicates that the Target wishes to perform a response cell transfer to the Initiator. The response cell is transferred when both the **RSPVAL** and **RSPACK** signals are asserted.
Response Content		
RDATA [8b-1:0] Response Data	TI MA	Response data is the data that is returned by the Target to the Initiator with Read-operations.
REOP Response End Of Packet	TI MA	Response End of Packet is asserted on the last cell of the response packet indicating that all cells associated with the given response packet have been transferred.
RERROR Response Error	TI	Response Error is asserted by the Target during a response packet to indicate that an error has occurred during the current packet.

Table 5: Signal List (continued)

4.3.5 System Level Signals

Clock

Signal Name:	Clock
Signal Abbreviation:	CLOCK
Polarity:	Active at Positive edge
Driven By:	System
Received By:	VCI Initiator, VCI Target

This signal provides the timing for the Virtual Component Interface and is an input to both the initiator and target that are connected via the BVCI. All initiator and target output signals are asserted/de-asserted relative to the rising edge of **CLOCK** and all initiator and target inputs are sampled relative to this edge.

Reset

Signal Name:	Reset
Signal Abbreviation:	RESETN
Polarity:	Asserted Negative

Driven By:	System
Received By:	VCI Initiator. VCI Target
Timing:	Asserted > 8 clock cycles

This signal is used during power-on reset and is used to bring the BVCI to an idle or quiescent state. This idle state is defined as the BVCI state in which:

1. The [CMD|RSP]VAL signals are de-asserted
2. The [CMD|RSP]ACK signals are de-asserted

The system must guarantee that **RESETN** is asserted for at least eight cycles of **CLOCK**.
(Larger if parameter RESETLEN is set.)

4.3.6 Request Signals

Command Valid

Signal Name:	CommandValid
Signal Abbreviation:	CMDVAL
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK == 1 and next rising edge of CLOCK.

The **CMDVAL** signal is driven by a VCI initiator to indicate that there is a valid request cell on the BVCI. All of the initiator request signals are qualified by **CMDVAL**. The initiator keeps **CMDVAL** asserted, and all of its control signals valid and stable, until it receives the **CMDACK** signal from the target. The initiator should not assert **CMDVAL** unless the current transaction is intended for the target. Thus, the initiator may need to perform address decoding on its on-chip bus side to generate **CMDVAL** for the target, thereby accomplishing device selection.

Command Acknowledge

Signal Name:	CommandAcknowledge
Signal Abbreviation:	CMDACK
Polarity:	Asserted Positive
Driven By:	VCI Target
Received By:	VCI Initiator
Timing:	Asserted and negated at rising edge of CLOCK

The **CMDACK** signal is asserted by the target to indicate the completion of a request between the initiator and target. In the case of write operations, this means that the target has accepted the data which is on the write data bus or will do so at he end of the current clock cycle. In the case of read operations, the assertion of the **CMDACK** by the target indicates that the target has accepted the request and started processing it. The request completes as soon as the rising edge of **CLOCK** samples **CMDACK**. The target may de-assert the **CMDACK** by the next rising edge of **CLOCK** unless a new command has been initiated by the initiator, or default acknowledge is used. Similarly, the initiator de-asserts **CMDVAL** by the next rising edge of **CLOCK** unless it is presenting a new request.

Command

Signal Name:	Command
Signal Abbreviation:	CMD

Polarity:	N/A
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK == 1 at rising edge of CLOCK.

This signal is a two-bit command-field asserted by the initiator and indicates the nature of the requested transfer. It is encoded as follows:

- 00b = NOP, no data is actually transferred (optional)
- 01b = READ, data is requested by the Initiator from the Target
- 10b = WRITE, data is transferred from the Initiator to the Target
- 11b = LOCKED READ, data is requested by the Initiator from the Target with the added function of locking out access to at least the particular cell until a write-operation is performed to the same cell-address

This signal must be valid any time that the **CMDVAL** signal is asserted.

End-of-Packet

Signal Name:	End-of-Packet
Signal Abbreviation:	EOP
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

The **EOP** signal is de-asserted by the initiator to indicate that the transfer being performed will be followed with a transfer by the initiator to the next higher cell address. This signal is used by the target device to pre-calculate address in order to improve the data transfer performance. The packet transfer is completed once a cell is transferred with the **EOP** signal asserted.

Chain Fixed

Signal Name:	ChainFixed
Signal Abbreviation:	CFIXED
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

Chain Fixed indicates that the opcode (**CMD**, **CONTIG**, **WRAP**, and **CONST**) and **PLEN** fields will be constant across the chain and the address field behavior will be identical among packets within a chain.

Chain Length

Signal Name:	ChainLength
Signal Abbreviation:	CLEN
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

Chain Length indicates the number of packets remaining in a chain. The last packet transferred in a chain will have a zero **CLEN** value. The **CLEN** value can also be tied off to zero if packet chaining is not required.

Contiguous

Signal Name:	Contiguous
Signal Abbreviation:	CONTIG
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

Contiguous-signal indicates that the sequence of addresses that will be accessed within the packet is contiguous. When **CONTIG** is asserted, the address is normally increased by the cell size in bytes. If a packet does not start or end at a cell boundary, the first or last transfer contains less bytes than a cell. In such cases, the address is increased not by the length of a cell but by the amount of bytes actually transferred in the first or the last cell. In middle-part of the packet, the address always increases by cell size, when **CONTIG** is active.

Packet Length

Signal Name:	PacketLength
Signal Abbreviation:	PLEN
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

Packet Length indicates the length of the packet in bytes. The valid range for **PLEN** is 1 to 2^k-1 (1 to 511 given that k is limited to 9). A value of 0 for **PLEN** can be used to indicate that the packet length is undefined (no implied packet length). In this case, the packet ends as a cell is transferred with **EOP** active. **PLEN** is held constant over the packet. Thus, **PLEN** does not indicate "remaining length".

Constant

Signal Name:	Constant
Signal Abbreviation:	CONST
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

Constant indicates that the address will remain constant throughout the entire packet. When **CONST** is asserted, **CONTIG** and **WRAP** are ignored.

Wrap

Signal Name:	Wrap
Signal Abbreviation:	WRAP

Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

Wrap is used in conjunction with **CONTIG** to indicate how addresses that increment past the boundary indicated by **PLEN** are handled. If **WRAP** is asserted and **PLEN** has only a single bit set (indicating a power of 2 packet size!), the address will wrap around. If the packet length is not power of two or if **CONTIG** is not active, the **WRAP** signal is don't-care. The wrap address equals to (**ADDR** and not (**PLEN**-1)) + **PLEN**. The next address after wrapping equals to (**ADDRESS** and not (**PLEN**-1)).

Address

Signal Name:	Address
Signal Abbreviation:	ADDRESS[n-1:0]
Polarity:	N/A
Driven By:	VCI Initiator
Received By:	VCI
TargetTiming:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

ADDRESS is the address of the request generated by the Initiator and received by the Target. The address updates for every cell transferred within a packet and must remain within the address space of a single Target. The pattern of addresses that are permissible are defined by the operation type (**CONTIG**, **WRAP**, and **CONST** signals). **ADDRESS** contains the lowest byte address for the first transfer in the packet. For all cells after the first transfer, **ADDRESS** is aligned to a cell boundary. The combination of a cell-aligned address and byte enables is sufficient to perform the transfer correctly. However, the addition of extra information in the first address may allow performance advantage in some systems. Note that a non cell-aligned address is endian dependent.

Byte Enable

Signal Name:	Byte Enable
Signal Abbreviation:	BE[**b**-1:0|0:**b**-1]
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

BE is a **b**-bit field that indicates which bytes of the cell being transferred are enabled. The **b** equals to total data width of the BVCI / 8. These signals must be valid any time that the **CMDVAL** signal is asserted. In write transfers, the disabled bytes are not overwritten. In read transfers, the target may or may not present the disabled bytes in the **RDATA** bus. They are ignored by the initiator. The direction of the **BE**-signal range numbering depends of the endiannes.

Write Data

Signal Name:	Write Data
Signal Abbreviation:	WDATA[8**b**-1 : 0]
Polarity:	N/A
Driven By:	VCI Initiator
Received By:	VCI Target

Timing:	Asserted at rising edge of CLOCK until CMDACK = 1 at rising edge of CLOCK.

The write data lines are driven by the VCI initiator and are used to transfer write data from an initiator to a target device. Write data consists of **b** logical byte lanes, based upon the capabilities of the target and is defined and fixed at the time of component instantiation. Allowed values of **b** are power of two. Bit **8*b-1** is the most significant bit of the most significant byte and bit 0 is the least significant bit of the least significant byte. The write data lines must contain valid write data while the **CMDVAL** signal is asserted and the **CMD** is indicating a write transfer.

For VCs supporting a data size that is not an eight bit increment, the next larger supported bus size will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide VCI with the 4 Most Significant bits tied to logic zero.

4.3.7 Response Signals

Response Valid

Signal Name:	ResponseValid
Signal Abbreviation:	RSPVAL
Polarity:	Asserted Positive
Driven By:	VCI Target
Received By:	VCI Initiator
Timing:	Asserted at rising edge of CLOCK until RSPACK = 1 at rising edge of CLOCK.

The **RSPVAL** signal is driven by a VCI target to indicate that there is a valid response on the BVCI. All of the target response signals are qualified by **RSPVAL**. The target keeps **RSPVAL** asserted, and all of its control signals valid and stable, until it receives the **RSPACK** signal from the target.

Response Acknowledge

Signal Name:	ResponseAcknowledge
Signal Abbreviation:	RSPACK
Polarity:	Asserted Positive
Driven By:	VCI Initiator
Received By:	VCI Target
Timing:	Asserted and negated at rising edge of CLOCK

The **RSPACK** signal is asserted by the target to indicate the completion of a response transfer between the initiator and target. This means that the initiator has accepted the response data, which is on the target's response signals, or will do so at he end of the current clock cycle. The response completes as soon as the rising edge of **CLOCK** samples **RSPACK**. The initiator may de-assert the **RSPACK** by the next rising edge of **CLOCK** unless a new response has been initiated by the target, or default acknowledge is used.

Read Data

Signal Name:	Read Data
Signal Abbreviation:	RDATA[8b-1: 0]
Polarity:	N/A
Driven By:	VCI Target
Received By:	VCI Initiator
Timing:	Asserted at rising edge of CLOCK until RSPACK = 1 at rising edge of CLOCK.

The read data lines are driven by the VCI initiator and are used to transfer read data from a target to an initiator device. Read data consists of **b** logical byte lanes, based upon the capabilities of the target and is defined and fixed at the time of component instantiation. Allowed values of **b** are power of 2 bytes. Bit 8***b**-1 is the most significant bit of the most significant byte and bit 0 is the least significant bit of the least significant byte. The read data lines must contain valid read data while the **RSPVAL** signal is asserted and the response is to a read request.

For VCs supporting a data size that is not an eight bit increment, the next larger supported bus size will be used with the unused bits tied to logic zero. For example, a 12-bit device must use a 16-bit wide BVCI with the 4 Most Significant bits tied to logic zero.

Response End-of-Packet

Signal Name:	ResponseEndofPacket
Signal Abbreviation:	REOP
Polarity:	Asserted Positive
Driven By:	VCI Target
Received By:	VCI Initiator
Timing:	Asserted at rising edge of CLOCK until RSPACK = 1 at rising edge of CLOCK.

Response End of Packet is asserted on the last cell of the response packet indicating that all cells associated with the given response packet have been transferred.

Response Error

Signal Name:	Response Error
Signal Abbreviation:	RERROR[E:0]
Polarity:	Positive
Driven By:	VCI Target
Received By:	VCI Initiator
Timing:	Asserted at rising edge of CLOCK when RSPVAL = 1, until RSPACK = 1 at rising edge of CLOCK.

Error signal is valid only when RSPVAL=1, with the following meaning:
For ERRLEN = 0 (E=0)
RERROR=0: Normal (no error)
RERROR=1: General data error. The entire packet is considered bad.

For ERRLEN = 1 (E=1)
RERROR = 00: Normal (no error)
RERROR = 01: General data error. The entire packet is considered bad
RERROR = 10: Reserved
RERROR = 11: Abort Disconnect

For ERRLEN = 2 (E=2)
RERROR =000: Normal (no error)
RERROR = xx0: Reserved
RERROR = 001: General data error. The entire packet is considered bad
RERROR = 011: Reserved
RERROR = 101: Bad data (retry)
RERROR = 111: Abort Disconnect

After receiving an error, the Initiator may or may not continue with the current packet. If it chooses to end the packet prematurely, it can do so by asserting EOP regardless of the PLEN value. After this, it can choose not to try a transfer anymore (Abort) or Retry part or all of the transfer. For any error, the Target must process the subsequent pending cells and packets with the normal protocol, i.e. it must continue sending responses until it has processed the EOP with or without further errors signaled. In general, the RERROR is more informative than prescriptive, and the Target may not assume any special behavior from the Initiator. The Initiator is anyhow encouraged to act responsibly, when it meets an error.

It is strongly recommended that all BVCI components would support at least one bit error. The BVCI target is strongly recommended to signal an error when it receives a request it does not support. Obviously, VCI provides also with a possibility to get error records through normal Read-requests from the Target, but this belongs to domain of a particular Virtual Component's implementation.

4.4 Basic VCI Protocol

4.4.1 Protocol Fundamentals

The Virtual Component Interface (VCI) protocol is described as a set of three stacked layers: transaction layer, packet layer and cell layer.

4.4.1.1 Transaction Layer

The transaction layer is above the concerns of hardware implementation. It defines the system as a series of communicating objects which can be either hardware or software modules. The information exchanged between Initiator and Target nodes is in the form of a request-response pair as illustrated in Figure 15 below.

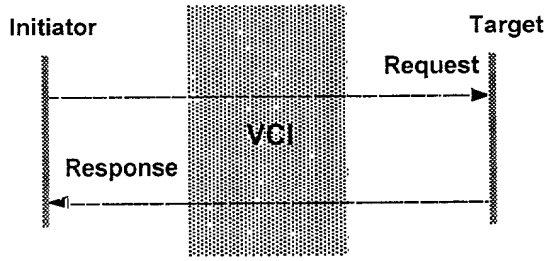

Figure 15. System Transaction Layer View of Information Transfer over the VCI

Transaction

A pair of request and response transfers is called a *transaction.* Typically, the basic unit of information exchanged is some form of data structure. As the communication is decomposed to lower layers of abstractions, the basic transfer unit becomes smaller.

4.4.1.2 Packet Layer

The packet layer adds generic hardware constraints to the system model. In this layer, VCI is a bus-independent interface that supports point to point physically address-mapped split transactions between Initiators and Targets in unit time. There will not yet be a commitment to a particular interface width. In this layer, the request and response information to be transferred across the interface is split into more manageable chunks, on the basis of generic hardware constraints. This is illustrated in Figure 16 below.

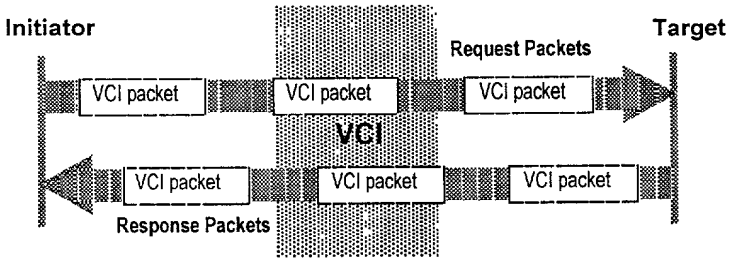

Figure 16. Packet Layer View of Information Transfer over the VCI

Operation

A transaction is called a *VCI operation* if the information is exchanged using atomic request and response transfers across the interface. The information exchanged during an operation is in the form of a VCI

packet, a packet layer transfer unit defined in the following section. In a packet layer, a VCI transaction decomposes into one or more operations.

Packet

Packet is the basic unit of information that can be exchanged over the VCI in an atomic manner. The point to point connection between Initiator and Target is maintained throughout the transfer of a packet.

The request and response transfer units of a VCI transaction are decomposed into one or more request-response packet pairs. Multiple packets can be combined to form larger, non-atomic transfer units called packet chains, as explained in the next chapter. A VCI operation is a single request-response packet pair. For each request packet transferred from Initiator to Target, there will be one corresponding response packet transferred from Target to Initiator. The Targets shall return response packets in the same order as request packets are issued.

The content of a packet depends on whether it is a request or response packet and the type of operation being carried out - such as read, write etc. The data field of request packets is relevant only for write operations. The field, **RDATA**, in response packets contains valid values only for read and locked_read operations. The request packet header contains information on packet chaining. More details on other packet fields will be introduced in subsequent sections.

Packet length is the number of bytes transferred during a read, write, or locked_read operation. This field is irrelevant for NOP operations. A zero value specified for **PLEN** in read, write, or locked_read operations indicates that the length of the packet is undefined. Long packets can result in reduced arbitration overhead and some optimization in certain areas like read pre-fetching and SDRAM access. This can improve the data transfer rate for that particular thread. However, very long packets transferred over a shared interconnect system lock out all other agents, causing degradation of the system performance. Hence, it is highly recommended to use short packets to optimize the overall system performance and to use the packet chaining mechanism, described in the next section, to create long, but non-atomic, transfers.

Packet Chain

The packet chain mechanism allows a VCI Initiator to describe linkage between related packets to the system. The system can combine chained packets into larger, more efficient, higher performance transfers that satisfy the constraints of the system application. Packet chains produce the same transfer efficiency that long packets provide without requiring the entire transfer to be atomic.

Two fields in the request packet header, ***CLEN*** and ***CFIXED,*** specify the packet chaining information. ***CLEN*** describes the number of packets remaining in the chain (not including the current packet). This field is normally decremented with every transmitted packet.

The flag, ***CFIXED,*** provides information about the linkage between the header fields among the packets in a packet chain. If set, ***CFIXED*** guarantees that the opcode and packet length fields will be constant across the chain.

The packet chain mechanism allows the VCI Initiators to minimize their packet length to that which is required for proper operation (i.e. functionality). This mechanism concurrently provides the interconnect logic and the Target with enough information for optimizing the chained transfers to meet application performance and efficiency requirements.

4.4.1.3 Cell Layer

The cell layer adds more hardware details such as interface width, handshake scheme, wiring constraints, and a clock to the system model. This layer is not concerned about shared interconnects and arbitration for such services. In the cell layer, VCI is viewed as a bus-independent, point to point interface that supports physically address-mapped split transactions between Initiators and Targets, using a cycle-based handshake

protocol. Introduction of interface width information enables decomposing of packets (the basic transfer unit defined in the packet layer) into cells that are handshaken under a cycle-based protocol across the definitive sized interface.

Cell

A cell is the basic unit of information, transferred on rising **CLOCK** edges under the **VAL-ACK** handshake protocol, defined by the cell layer. Multiple cells constitute a packet. Both request and response packets are transferred as series of cells on the VCI. The number of cells in a packet depends on the packet length and the interface width.

The structure of a cell is very similar to that of a packet, except for the decomposing of **WDATA** and byte enable fields and the introduction of end-of-packet fields. The opcode fields, chaining information, and **PLEN** in a request cell are the same as the corresponding request packet fields.

The request cell structure and response cell structure is detailed in Tables 6 and 7 below.

Comment		Field	Description
Packet chain Information		**CLEN**	Number of remaining packets in the chain
		CFIXED	Indicates if opcode and **PLEN** are same across all packets in the chain
		ADDR	Address of each cell being transferred
		PLEN	Packet length: Total number of data bytes transfered in the operation
Opcode		**CMD**	Command: read, write, nop or locked_read
	Flags	**CONTIG**	Contiguous address mode
		WRAP	**WRAP** address mode
		CONST	**CONST** address mode
		BE	Byte Enable: indicates which bytes are involved in the operation
Optional field		**WDATA**	Data in write requests
		EOP	Indicates if the cell is the last one in the request packet

Table 6: Request Cell Structure

Comment	Field	Description
	RERROR	Error status for the cell
Optional field	**RDATA**	Data in read/locked_read responses
	REOP	Indicates if the cell is the last one in the response packet

Table 7: Response Cell Structure

VAL-ACK Handshake

The **VAL-ACK** handshake provides unambiguous synchronization of Initiator and Target modules for transferring cells over the interface. It is a simple handshake protocol based on two control signals. Two separate sets of handshake signals are used in the VCI interface: one for transferring request cells from Initiator to Target, and the other for transferring response cells from Target to Initiator. The transfer of request and response cells over the interface are completely de-coupled, concurrent events. The handshake fundamentals explained below apply to both request and response channels. Generic names, **VAL** and **ACK**, are used in the discussion for the control signals. These names represent **CMDVAL** and **CMDACK** signals of the request channel and **RSPVAL** and **RSPACK** signals of the response channel.

Cell transfer on the channel is solely controlled by the following two signals (Table 8):

VAL	Driven by the initiator module to indicate that a cell was placed on the interface and is ready for transfer
ACK	Driven by the target module to indicate that it has received the cell, if any was present as

	announced by **VAL**.

Table 8: Handshake Signals

Table 9 summarizes different encoding for the **VAL-ACK** signals and the corresponding channel states.

VAL	ACK	State	Description
0	0	IDLE	The channel is in idle state.
1	0	VALID	Valid is asserted. Waiting for Acknowledge.
0	1	DEFAULT_ACK	Ready to Grant
1	1	SYNC	Handshake synchronization

Table 9: VAL-ACK Encoding and Channel States

The channel is in IDLE state when both **VAL** and **ACK** are de-asserted. The initiator module unconditionally asserts **VAL** when the cell information is placed on the interface. The target module may delay the assertion of **ACK** if it is not ready to accept the cell. The channel is said to be in REQUEST state if **ACK** is not asserted for a **VAL** on the rising edge of the **CLOCK**. The channel state transitions to SYNC when both **VAL** and **ACK** are asserted. The handshake synchronization and cell transfer occur on each **CLOCK** edge when the VCI is in SYNC state. (See Figure 17.)

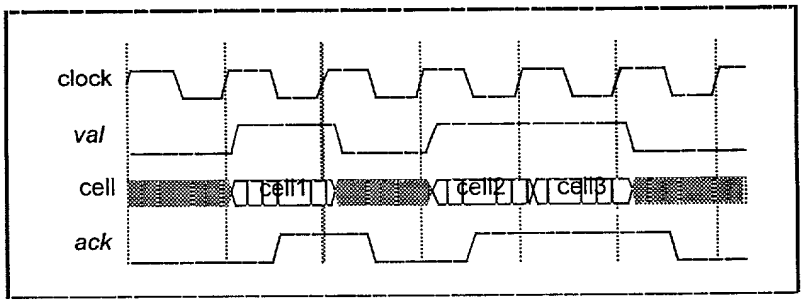

Figure 17. VAL-ACK Handshake for Single-Cell Transfer

While transferring a multi-cell packet, either module can insert wait cycles by de-asserting **VAL** or **ACK**. The receiving module can assert **ACK**, even when **VAL** is not present, to indicate that it is ready to **ACK** the next **VAL**. The corresponding channel state is DEFAULT_ACK. In this case, cell transfer will occur on the first **CLOCK** edge on which **VAL** is asserted.

Once the initiator module asserts **VAL**, it cannot change that signal until the requested cell is transferred, regardless of the state of **ACK**. Similarly, once the receiving module asserts **ACK**, it should not de-assert that signal until a cell is transferred. Generally, neither module shall change its mind once it has committed to the cell transfer.

However, de-committing of the **ACK** signal may be necessary in certain scenarios, such as a default **ACK** from a bus wrapper module to an Initiator, where this bus is "parked". Such deviations from the recommended interface behavior should be clearly documented when VCs are implemented. Table 10 below summarizes all the permitted state transitions.

From	To	Validity	Description
IDLE	IDLE	Valid	Continues in idle
IDLE	VAL	Valid	New Valid asserted
IDLE	DEFAULT_ACK	Valid	Default Acknowledge asserted

IDLE	SYNC	Valid	New Valid acknowledged on same **CLOCK**
VAL	IDLE	Invalid	Valid de-committing not allowed
VAL	VALID	Valid	Valid maintained
VAL	DEFAULT_ACK	Invalid	Valid de-committing not allowed
VAL	SYNC	Valid	Valid Acknowledged
DEFAULT_ACK	IDLE	Invalid	Acknowledge de-committing not allowed
DEFAULT_ACK	VAL	Invalid	Acknowledge de-committing not allowed
DEFAULT_ACK	DEFAULT_GRANT	Valid	Acknowledge maintained
DEFAULT_ACK	SYNC	Valid	Valid Acknowledged
SYNC	IDLE	Valid	Back to idle
SYNC	VAL	Valid	New Valid asserted
SYNC	DEFAULT_ACK	Valid	Default Acknowledge asserted
SYNC	SYNC	Valid	Consecutive Valid-Acknowledge

Table 10: Permitted State Transitions

VAL Time and ACK Time

VAL_time and ACK_time are introduced as a way of classifying the handshakes. Definitions follow:

- VAL_time = number of cycles where **VAL** is de-asserted between cell transfers.
- ACK_time = number of cycles where **VAL** is asserted before **ACK** is asserted.

Figures 17 – 19 illustrate different **VAL-ACK** handshakes.

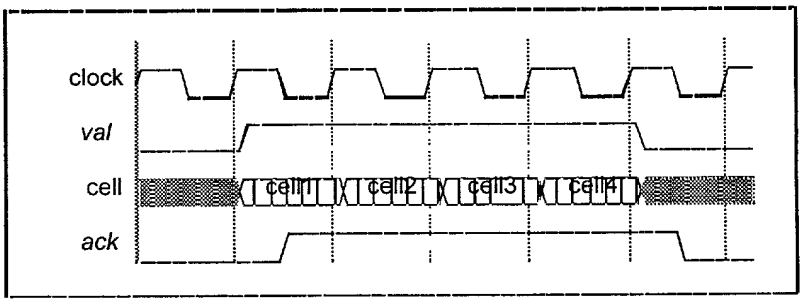

Figure 18. VAL-ACK Handshake with VAL Time = 0, ACK Time = 0

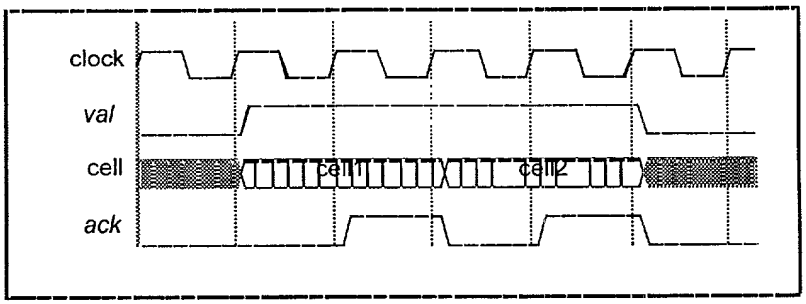

Figure 19. VAL-ACK Handshake with VAL Time = 0, ACK Time = 1

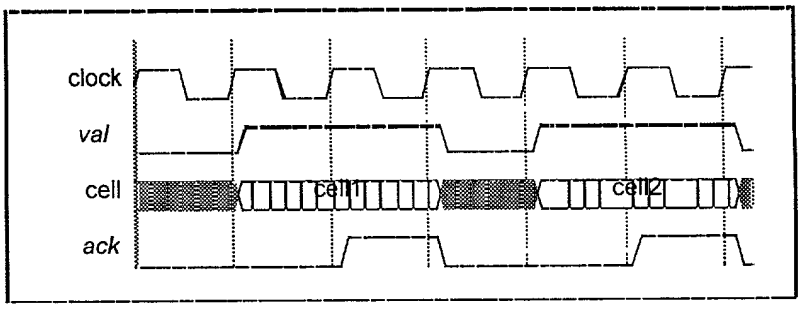

Figure 20. VAL-ACK Handshake with VAL Time = 1, ACK Time = 1

4.4.2 Basic VCI Operations

The basic transfer mechanism in VCI is packet transfer. Every operation on the VCI consists of a request packet transfer from the Initiator to the Target and a response packet in return. Both Initiator and Target modules are responsible for transferring the VCI request and response packets across the interface in an atomic manner. A packet is sent as a series of cells with the **EOP** (end of packet) field in the last cell set to value 1. Each cell is individually handshaken across the interface under the **VAL-ACK** handshake. Either the Initiator or the Target can insert wait cycles between cell transfers by de-asserting **VAL** or **ACK**. The order of packets and the number and order of cells within packets should be maintained the same between request and response transfers.

The transfer of request and response cells over the interface are completely de-coupled concurrent events. The signals used for encoding cell information, handshake signals, and the timing of the request and response transfers are totally separate.

All VCI signals are sampled on the rising edge of the **CLOCK**. The handshake signals **CMDVAL**, **CMDACK**, **RSPVAL**, and **RSPACK** are sampled on every rising **CLOCK** edge. The rest of those signals that encode the request and response cells are sampled on rising **CLOCK** edges, qualified by the corresponding **VAL** signal. The timing diagrams in this section show the relationship of relevant signals involved in illustrated operations.

4.4.2.1 Read Operation

Figure 21 below illustrates a 32-byte read operation on a 32-bit wide VCI. The operation starts with the Initiator placing the first cell of a request packet on the interface and asserting **CMDVAL** on **CLOCK** 2. The cell information is encoded in the signals: **CMD**, **CONTIG**, **WRAP**, **CONST**, **PLEN**, **EOP**, **ADDRESS**, and **BE**. The fields on chaining information, **CLEN** and **CFIXED**, are both set to value 0 for this operation and are not shown in the diagram.

As shown in Figure 21:

- The **CMD** and addressing mode flags, **CONTIG**, **WRAP**, and **CONST**, specify that the packet is part of a read operation with contiguous cell addresses.
- The field **PLEN** indicates that the packet length is 32 bytes.
- The address field contains the address of the Target and the byte address of the location from where the data is requested.
- **BE** indicates which byte lanes are involved in the first cell transfer.
- **EOP** is set to 0 to indicate that the current cell is not the last one in the packet.

The earliest the cell transfer can complete is in **CLOCK** period 2 itself, if the signal **CMDACK** is asserted combinatorially or if the channel is in default acknowledge state. In this example, the target module asserts **CMDACK** in the **CLOCK** period 3 (ACK time = 1) and the cell transfer completes in that **CLOCK** cycle.

The Initiator module updates the **ADDR** and **BE** signals to reflect the address and byte enable information for the second cell of the packet being transferred. Note that the address(n-1:m) field does not change. (It is illegal to address more than one Target within same packet.) The signals: **CMD, CONTIG, WRAP, CONST**, and **PLEN** (also **CLEN** and **CFIXED,** which are not shown) are common to all the cells in the request packet.

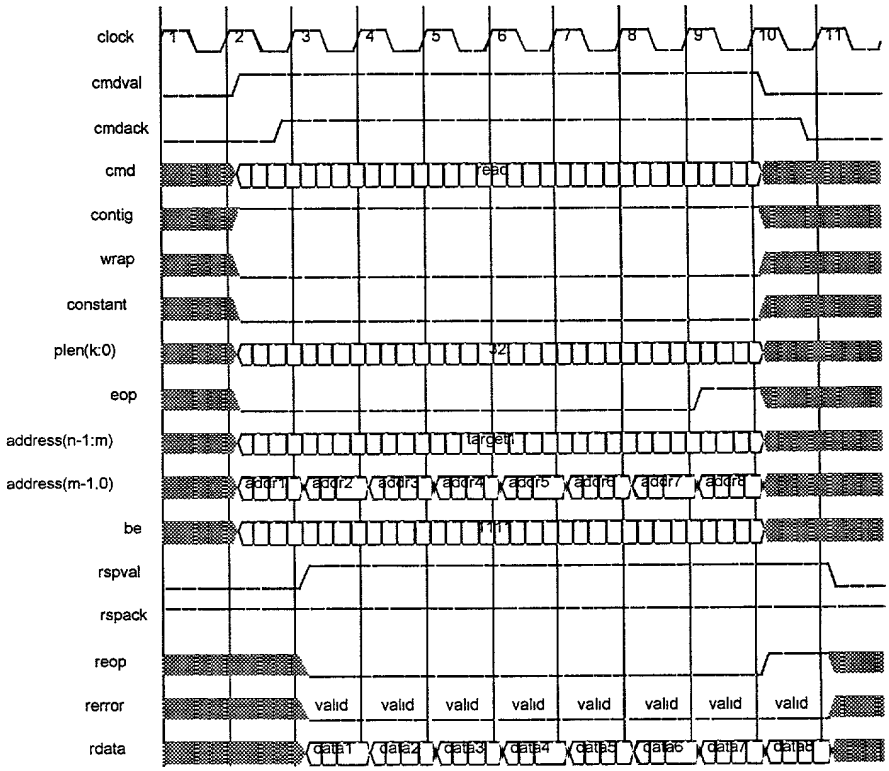

Figure 21. 32-byte Read Operation on a 32-bit VCI

All the eight cells in the request packet are transferred in 8 **CLOCK** cycles with **CMDVAL** and **CMDACK** asserted corresponding to VAL time = 0 and ACK time = 0. The **CMDVAL** for the last cell is asserted at **CLOCK** period 9 and the cell transfer completes on the same cycle. Note that the Initiator asserted the signal **EOP** at **CLOCK** 9 to indicate that the cell being transferred is the last one in the packet.

In the response channel, the Initiator continuously asserts the signal, **RSPACK**, indicating its readiness to accept the response cell with ACK time = 0. This is a default acknowledge condition, as the **CMDACK** is asserted when **CMDVAL** is not present. All the eight response cells are transferred in 8 **CLOCK**s as both VAL time and ACK time are 0. The first cell is transferred in **CLOCK** 3 when the Target module asserts **RSPVAL**. The Target continues to assert **RSPVAL** until **CLOCK** 10, transferring one cell on every **CLOCK** edge. At **CLOCK** 10, the Target also indicates to the Initiator through **REOP** that the current cell is the last one in the response packet. Each response cell contains the read data (on **RDATA**) and the error flag (on **RERROR**).

4.4.2.2 Write Operation

Figure 22 illustrates a 32-byte write operation on a 32-bit VCI. A write operation is similar to a read operation except that:

1. The Initiator provides the write data on signal **WDATA** as part of each request cell, and

2. The field **RDATA** is not significant.

The operation starts with the Initiator placing the first cell of a request packet on the interface and asserting **CMDVAL** on **CLOCK** 2. The cell information is encoded in the signals: **CMD, CONTIG, WRAP, CONST, PLEN, EOP, ADDRESS, BE,** and **WDATA.**

As shown in Figure 22 below:

- The signal, **CMD**, indicates that the packet is part of a write operation.
- The addressing mode flags, **CONTIG, WRAP,** and **CONST**, specify that the cell addresses are contiguous.
- **BE** indicates which byte lanes are valid in the write data.

The Initiator module updates the **ADDRESS, BE,** and **WDATA** signals upon every cell transfer to reflect the byte address, enable information, and the data, until all the four cells in the packet are transferred. Note that the address is updated for every cell, even though the opcode field indicates that the cell addresses are contiguous.

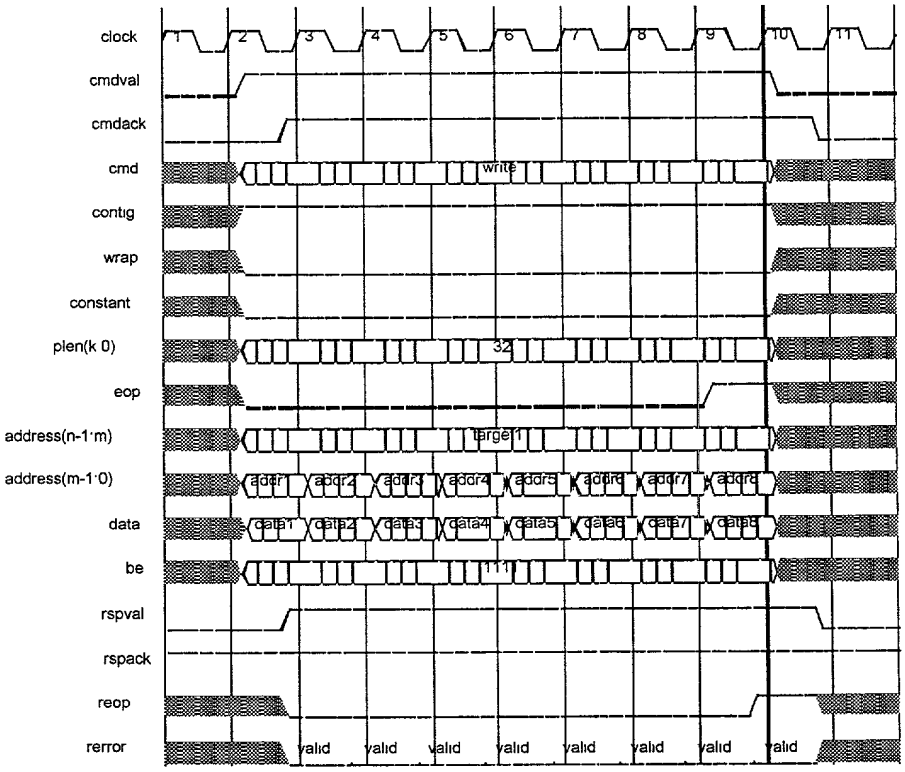

Figure 22. 32-byte Write Operation on a 32-bit VCI

All the eight cells in the request packet are transferred in 8 **CLOCK** cycles with **CMDVAL** and **CMDACK** asserted corresponding to VAL time = 0 and ACK time = 0. The response cells are also

transferred in 8 **CLOCK**s. The only information contained in write response cells is the error flag (on **RERROR**).

4.4.2.3 Other Operations

Two other operation types supported on VCI are NOP and Locked Read. The NOP operation is similar to a read or write operation, except that the **CMD** field of the request packet contains value '00' (NOP) and there is no data transferred along with request and response cells.

The locked read operation is similar to a read operation. The only difference is in the **CMD** signal encoding. The Target module/system locks out at least the memory locations addressed by the locked read request until another operation is issued to the same locations by the same Initiator.

Packet Chain Transfer

Figure 23 below illustrates a read transaction on the VCI composed of two operations grouped through the packet chaining mechanism. The operations are similar to the normal read operations except that the packet chaining information is provided along with the request packets. This information is encoded on signals **CLEN** and **CFIXED**.

CLEN specifies the number of packets remaining in the chain excluding the current one. In this example it is 2-1 = 1. Note that this information is **CONST** within the packet and dynamic across packets. **CLEN** is specified as 0 for the second request packet, indicating that it is the last packet in the chain.

The flag, **CFIXED**, is asserted for the first packet to indicate that same opcode will be used for all the packets in the chain. **CFIXED** also guarantees that the address relationship between the last cell of a packet in the chain and the first cell of the following packet will be the same as the one defined by the **ADDRESS** mode flags for cells within packets. In this example, **CFIXED** = 1, guarantees that the cell addresses will be contiguous across the packet chain.

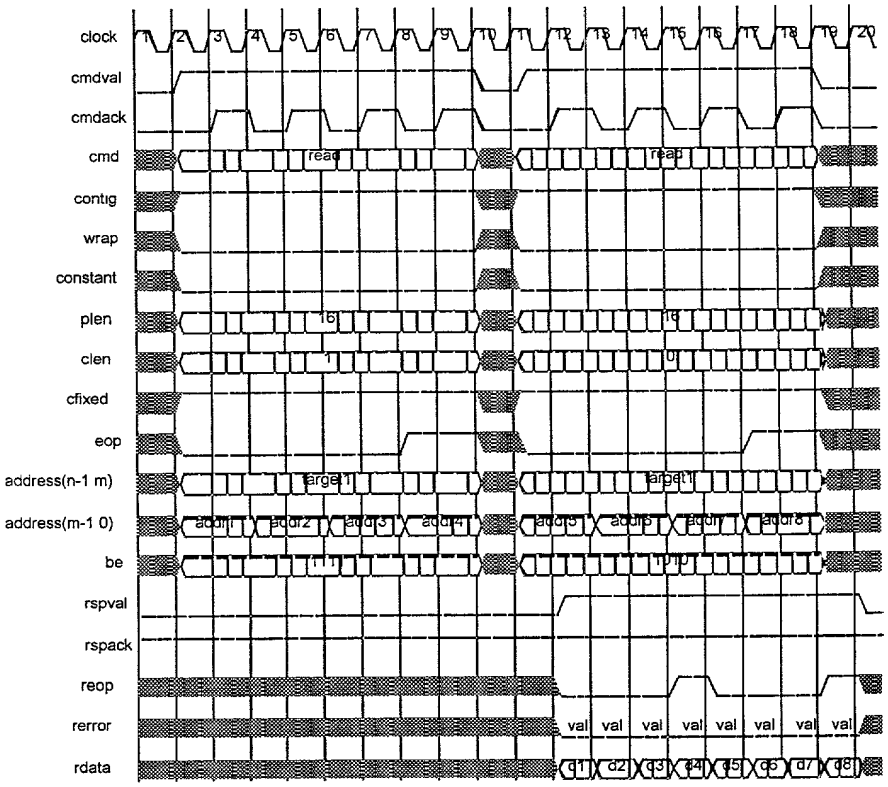

Figure 23. Packet Chain Transfer on the VCI

The chaining information on the first packet along with the packet header information conveys to the Target module that the Initiator module intends to read (**CLEN** + 1) x **PLEN** = (1 + 1) x 16 = 32 bytes of data from contiguous address locations starting from ADDR1. This information provides an option to the Target module to combine the two request packets to improve the transfer efficiency, if possible. In this example, it is assumed that the Target treats the packet chain as a single packet containing 8 cells to perform a high efficiency data fetch. Note that the first response packet has a latency of 9 **CLOCK**s whereas the second response packet arrives 4 **CLOCK**s after the arrival of the corresponding request packet.

4.4.2.4 Address Modes

The VCI requires the Initiator to provide cell addresses along with every request cell transferred to the Target module. In addition, the Initiator may also specify a predefined algorithm to calculate subsequent cell addresses from the previous cell address using the three addressing mode flags, **CONTIG**, **WRAP**, and **CONST**. When none of the flags are asserted by the Initiator, it indicates that there is no predefined relationship among cell addresses.

Random Address Mode

When none of the opcode flags are asserted, the initiator specifies random cell addressing mode. In this case, there are no predetermined relationships among cell addresses. Figure 24 below illustrates a 16-byte read operation with random address mode.

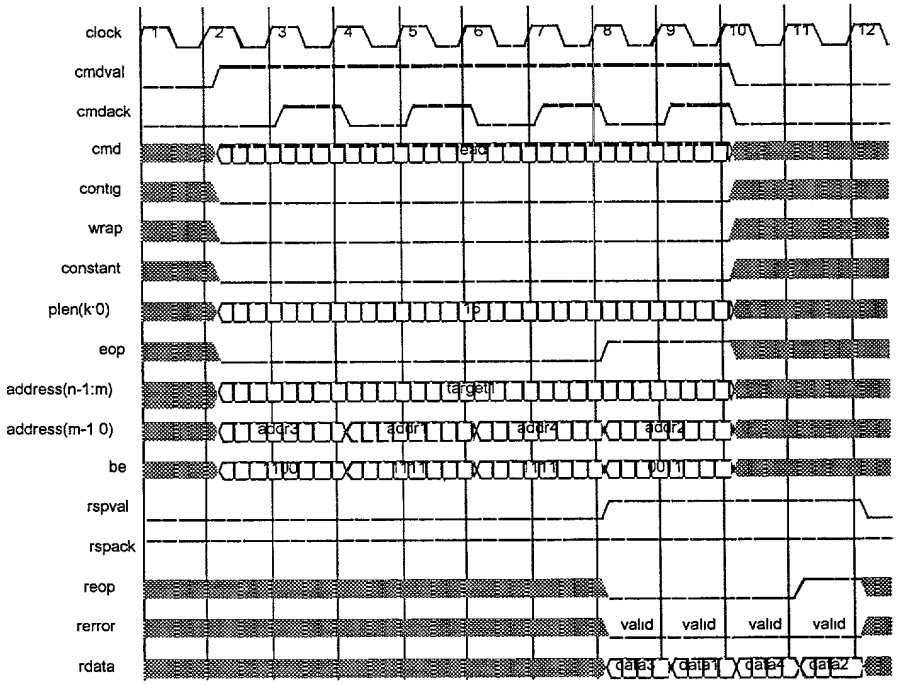

Figure 24. 16-byte Read Operation with Random Address Mode

Contiguous Address Mode

When **CONTIG** is asserted and **WRAP** is not asserted, the cell addresses advance in a contiguous manner. In this case, cell addresses can be calculated by adding the number of bytes transferred in the previous cell to the previous cell address. Figure 25 below illustrates a 12-byte read operation with contiguous address mode.

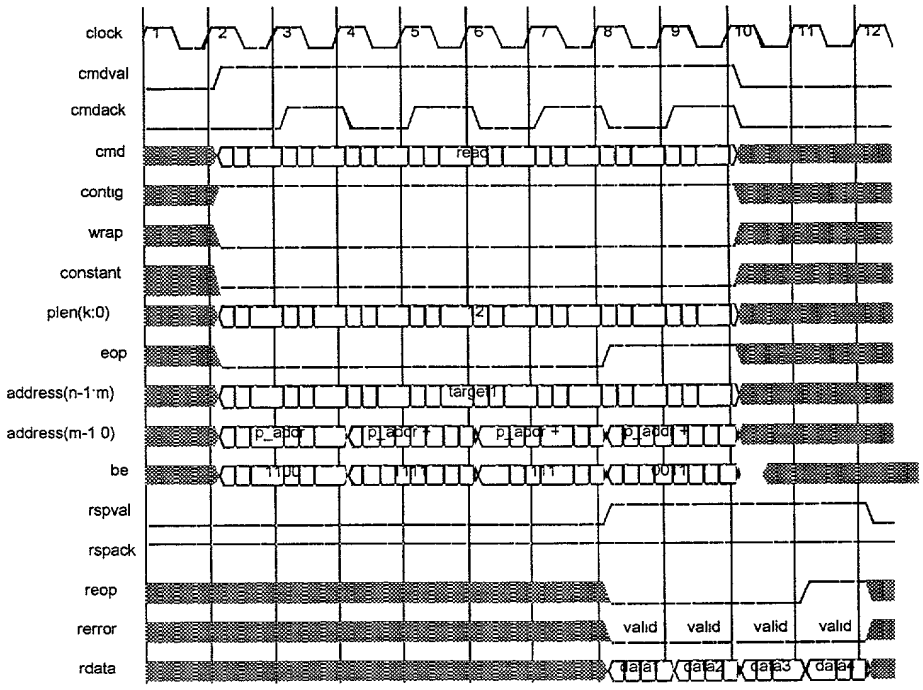

Figure 25. 12-byte Read Operation with Contiguous Address Mode

Wrap Address Mode

This addressing mode is specified by assertion of the flags, **CONTIG**, and **WRAP**. If the **WRAP** mode is specified and the field **PLEN** is specified as 2^A, then the following action take place. The cell address advances in a contiguous manner and wraps around when the sum of the number of bytes transferred and the lower A+1 bits of the cell address is more than the value of **PLEN**. (The carry bit does not propagate beyond bit location A.) A 16-byte read operation in **WRAP** address mode is illustrated below in Figure 26.

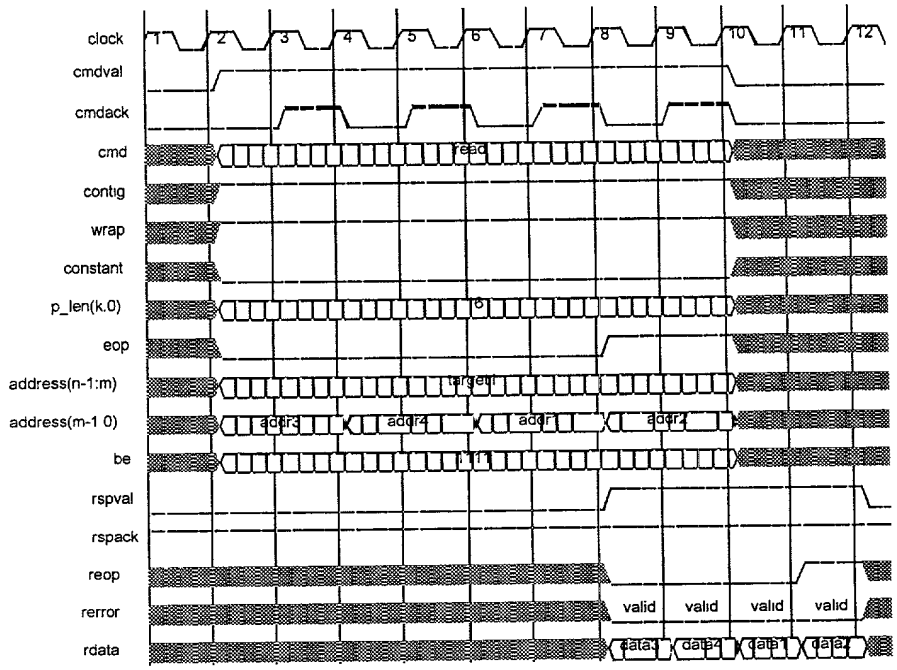

Figure 26. 16-byte Read Operation with Wrap Address Mode

Constant Address Mode

The constant address mode is specified using the flag **CONST**. When this mode is specified for a request packet, the cell address remains the same as the starting cell address across all the cells in the packet/packet chain. Figure 27 below illustrates a 16-byte read operation in constant mode. Note that all the bits in the **ADDRESS**(n-1.0) remain constant for all the cells in the packet.

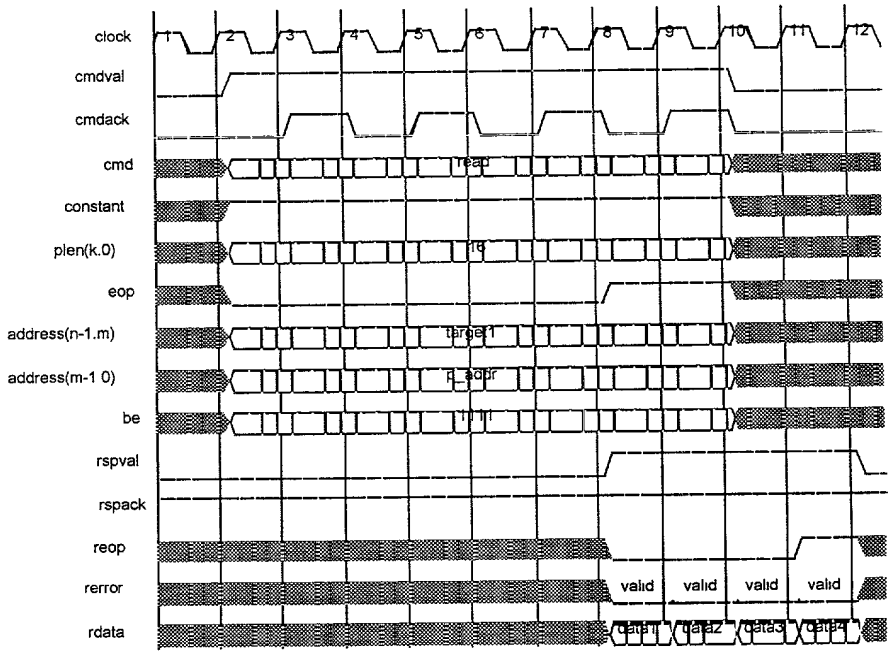

Figure 27. 16-byte Read Operation with Constant Address Mode

4.4.3 Basic VCI Signaling Rules

The following rules apply:

- For each packet transferred on the request channel, there should be exactly one response packet transferred back on the response channel.
- The order in which packets are transferred on the response channel should be same as the order in which the corresponding packets appear on the request channel.
- The number of cells in response packets should be same as in the corresponding request packets irrespective of the opcode.
- The order of cells within a response packet should be same as the order in the corresponding request packet.
- Once **CMDVAL** is asserted, the initiator module cannot change that signal until the requested cell is transferred regardless of the state of **CMDACK**.
- Once a cell is placed on the interface and **CMDVAL** is asserted, the initiator module cannot modify the cell content until the requested cell transfer is complete.
- **ADDRESS**(n-1:m), the top n-m bits of the **ADDRESS** signal which indicates the target being addressed, should remain constant across all the cells in a packet.
- Opcode signals, **CMD**(1:0) and the **ADDRESS** mode flags should remain constant throughout the packet transfer.
- **PLEN** signal, which indicates the length of the request packet, should remain constant throughout the packet transfer.
- **CLEN** and **CFIXED** signals should remain constant across all the cells within a packet. It is recommended that the field, **CLEN**, should either stay constant or decrement (by one) with every packet transfer. It is legal for the initiator to change **CLEN** in unexpected ways between packets. This behavior is discouraged, as it adds complexity to the Target. Initiators that exhibit this discouraged behavior must document it, and Targets must document what CLEN behavior they can accept.
- Changing the **CMD** field between packets in a packet chain is strongly discouraged. Initiators that exhibit this discouraged behavior must document it.
- Asserting the flag **WRAP** without asserting **CONTIG** is invalid. Asserting **WRAP** when **PLEN** is not a power of 2 value is also invalid.
- The Initiator shall not assert byte enable bits for those bytes that are outside the address range indicated by the address and **PLEN** fields.
- It is recommended that once the receiving module asserts **CMDACK**, the module cannot change that signal until the next clock cycle. This insures the transferred cell is not corrupted..

4.4.4 Additional Timing Diagrams

Additional timing diagrams follow.

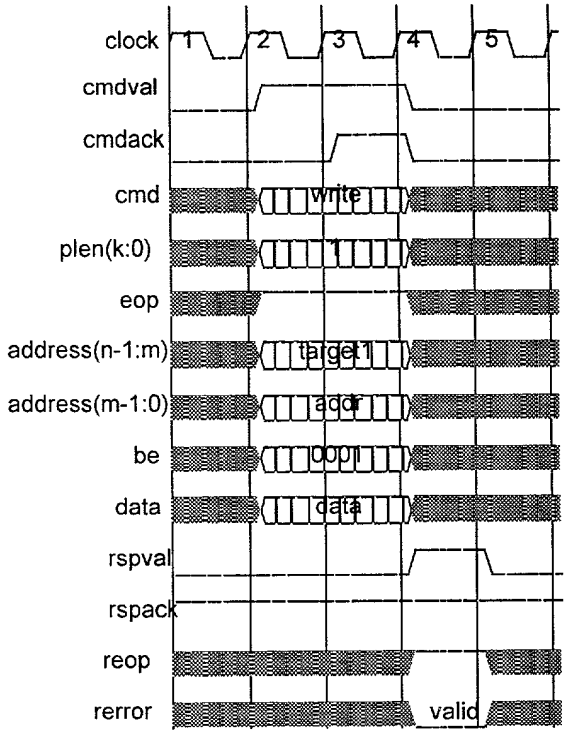

Figure 28. 1-byte Write Operation on a 32-bit VCI

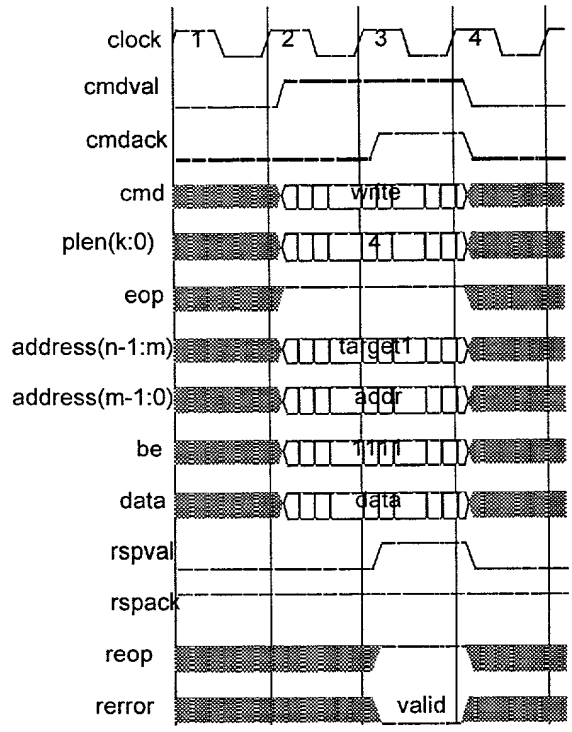

Figure 29. 4-byte Write Operation on a 32-bit VCI

5. Design Guidelines

5.1 User' Guide

This section describes the high-level responsibilities of the VC Initiator and VC Target, together with the associated implementation considerations, discussing also about the differences between Basic VCI and Peripheral VCI. The focus is especially in describing use of the VCI with an on-chip bus.

While the VC Initiator to OCB Initiator Wrapper interface is identical in protocol to the OCB Target Wrapper to VC Target interface, the differences between the VC and the OCB show up in the implementation considerations. There are at least two good reasons to keep the VC->OCB interface identical to the OCB->VC interface:

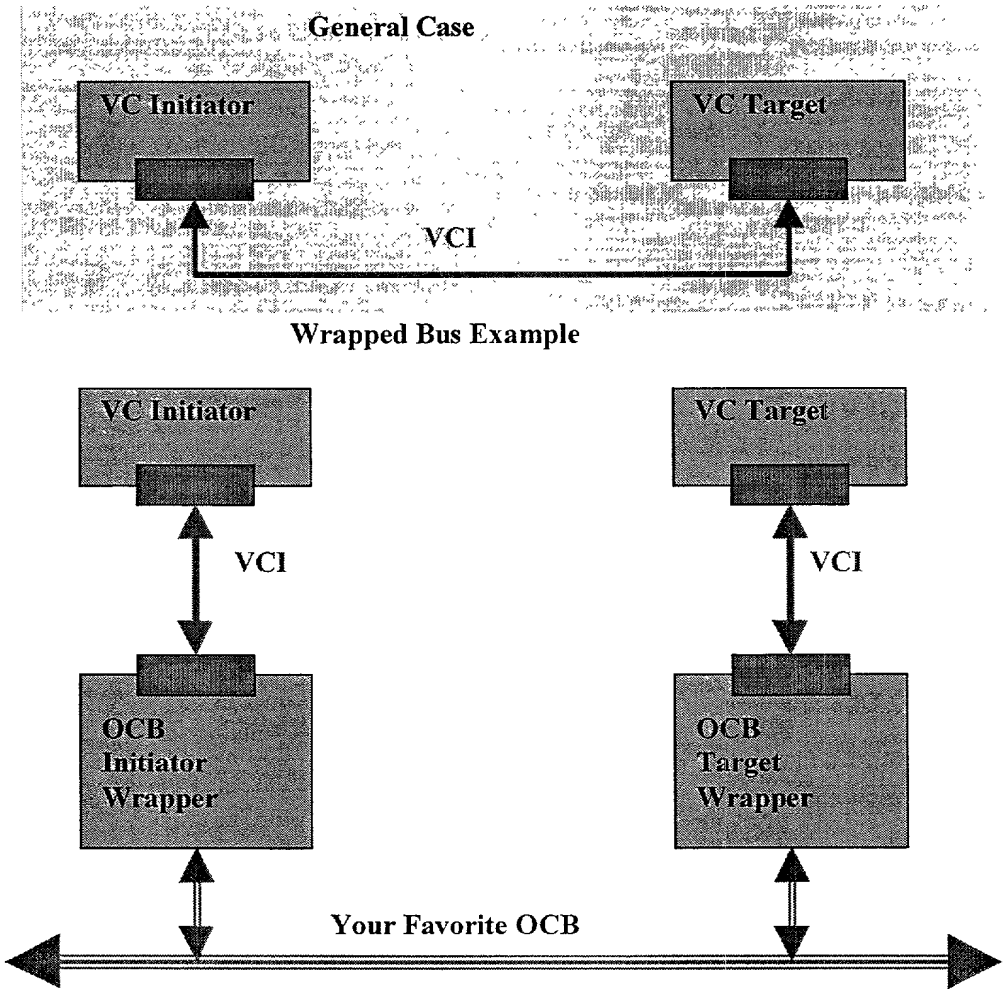

Figure 30. VCI Block Diagram

1. This facilitates direct VC=>VC connections (i.e. without an intervening OCB)

2. VC Initiators have communication features similar to OCBs (pretty obvious, given the heritage of most OCBs). Thus, we can use existing VC Initiators as reasonable test cases to ensure that the OCB

Initiator responsibilities are reasonable. A similar argument may be made for the OCB Target responsibilities.

The intended use for the Peripheral VCI is in VC-VC communication, and wrapping a VCI Target to an On-Chip Bus. Although it is possible to use the Peripheral VCI as a bus initiator, this is an unlikely case, since usually the peripheral bus initiator is a bus bridge from a system bus. So, the Peripheral VCI properties are tuned more towards the use as a peripheral bus target. The Basic VCI is more versatile and its features are tuned towards use as either an initiator or a target for an OCB.

Another possible topology is presented in Figure 31. In this case, there is no bus, but all the VCI Targets are connected point-to-point to a central OCB-VCI bridge. In this case, the bridge looks like a single target to the OCB in question, and contains a number of VCI Initiators. The bridge is more complicated than a bus wrapper, but the electrical design of point-to-point VCI connections may be easier than design of a shared bus.

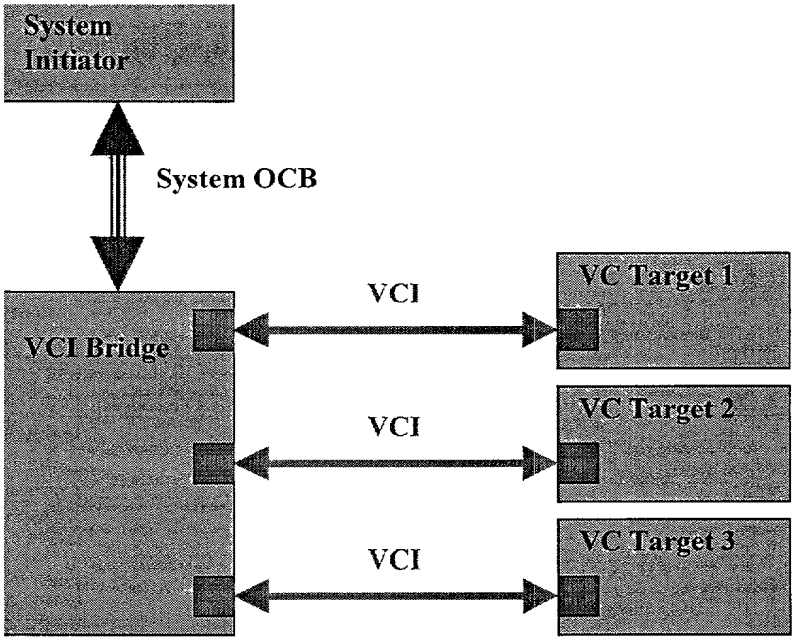

Figure 31. VCI with Star Topology

The following discussion covers the responsibilities of four different entities, as depicted in Figure 30: VC Initiator, OCB Initiator Wrapper, OCB Target Wrapper, and VC Target. Main differences with the BVCI and the PVCI are discussed.

5.1.1 VC Initiator Responsibilities

The VC Initiator is in complete control over the presentation of requests on its VC Interface; there is *no* arbitration. It asserts the CMDVAL (VAL for PVCI) signal to indicate that a valid request information are present. The size of ADDRESS and [W|R]DATA are dictated by the VC's capabilities, as is the set of supported transfer widths. It must hold the request information stable until it samples CMDACK (ACK for PVCI) asserted at the rising edge of CLOCK. On a read transfer, PVCI Initiator must acquire RDATA once it samples ACK asserted at the rising edge of CLOCK in PVCI. In BVCI, the VC Initiator can delay acquiring RDATA by holding RSPACK de-asserted, while RSPVAL is asserted. Once the BVCI Initiator asserts the RSPACK, it must complete the response transfer once it samples RSPVAL asserted at the rising edge of CLOCK.

The provider of the VC Initiator must provide a list of VCI configuration parameters for the VC, including such aspects as address/data bus widths, supported transfer widths, and timing data. These parameters allow the system integrator to configure the OCB Initiator Wrapper to meet the constraints of both the system and the VC Initiator.

5.1.2 OCB Initiator Wrapper Responsibilities

The OCB Initiator Wrapper is responsible for accepting the request from the VC Initiator, and controlling the OCB (as an OCB Initiator) to accomplish the transfer. In particular, the OCB Initiator Wrapper is responsible of making request to and accepting responses from the bus arbiter, initiating the transfer on the OCB (inserting any required OCB turn-around cycles), handling any OCB-level handshaking and getting Read data from the OCB. In most instances, the OCB Initiator Wrapper should not need to store any request information (particularly address or write data) on behalf of the VC Initiator, since the OCB Initiator Wrapper CMDACK (ACK) signal can be used to force the VC Initiator to hold the request information. It may need to store response information (such as RDATA) in the BVCI, if the VC Initiator expresses that it cannot accept the read data by not asserting the RSPACK.

However, electrical concerns will typically force the OCB Initiator Wrapper to buffer the request signals to drive long OCB wires, as well as dealing with OCB topology issues (tri-state vs. multiplexed buses, for instance). The OCB Initiator Wrapper is responsible for ensuring that the OCB timing is correct, and that VCI RDATA and handshaking signals make the required setup time to the VC Initiator.

The OCB Initiator Wrapper should be configurable enough to adapt to a range of possible VC Initiator capabilities. In particular, the OCB Initiator Wrapper should have variable address and data bus widths, and support for multiple transfer widths. This configurability allows the system integrator to match the OCB Initiator Wrapper to the VC Initiator.

Several issues arise with respect to the configuration of the OCB Initiator Wrapper. A VC Initiator with a 16-bit ADDRESS may need to connect to an OCB with a 32-bit address. In such a case, the system integrator should be free to synthesize the most significant 16 bits of the OCB address by whatever means are appropriate for the system. One way might be to statically configure the upper bits into the OCB Initiator Wrapper. Another method might involve placing a writable register somewhere on the OCB to hold the upper bits. The OCB Initiator Wrapper is also responsible for any required data shifting. Data shifting which can arise from endianness differences between the VC Initiator and the OCB, or width mismatches should not be required, in general. This requirement can generally be lifted by the appropriate connection of the VC byte lanes to the VCI byte lanes, if the endianness is of static nature.

5.1.3 OCB Target Wrapper Responsibilities

The OCB Target Wrapper is in complete control over the presentation of requests on its VCI; there is *no* arbitration. The OCB Target Wrapper acts as an OCB Target; it must convert the OCB transfer into a VCI-compliant transfer. In particular, it must translate OCB request information into the VCI CMDVAL (VAL) signal, which indicates the presence of a valid VCI request *that is intended for the VC Target.* Device selection mechanics are very specific to different OCBs. Traditional approaches to device selection include distributed address decoding and centralized address decoding. For a distributed-decoding OCB, the address decoder to determine VCI CMDVAL (VAL) will likely live inside the OCB Target Wrapper. For a centralized-decoding OCB, the VCI CMDVAL (VAL) signal is likely just a buffered version of the select signal from the centralized decoder.

The OCB Target Wrapper presents the request information across the VCI to the VC Target. It must hold the request information stable until it samples the VCI CMDACK (ACK) asserted. Since the OCB Target Wrapper is responsible for ensuring compliance with the OCB transfer protocol, most OCB Target Wrappers will use OCB Target handshaking to force the OCB Target Wrapper to hold the request information stable (i.e. insert wait states). Thus, data and address storage is rarely required in the OCB Target Wrapper. The OCB Target Wrapper may need to delay the assertion of CMDVAL so that it and the other request fields satisfy the VC Target setup time. It should wait until the VC Target returns CMDACK

before releasing the OCB resources associated with the transfer. It is the OCB Target Wrapper's responsibility to ensure that VCI RDATA makes the timing path back across the OCB to the OCB Target Wrapper. Since the two-wire Peripheral VCI handshake does not allow the OCB Target Wrapper to force the VC Target to hold RDATA, the OCB must run slow enough to allow RDATA to make it back across the OCB or else data storage is required inside the OCB Initiator to acquire RDATA. The Basic VCI handshake allows for the OCB Target Wrapper to stall the VC Target. In any case, it is typically the OCB Target Wrapper's responsibility to buffer RDATA to drive the OCB data wires, thereby isolating the VC Target from the loading and topology (tri-state, multiplexed, etc.) of the OCB.

The OCB Target Wrapper should be configurable enough to match its VCI address and data bus widths, and transfer widths to the VC Target's implementation of the VCI. This configurability is key to allow the system integrator to design working systems using the VCI. When the OCB has a different data bus width than the VC Target, it is the OCB Target Wrapper's responsibility to accomplish any required data multiplexing. The VC Target ADDRESS is typically much narrower than the OCB ADDRESS; the VC Target should be configurable so that only the required ADDRESS bits are presented to the VC Target.

5.1.4 VC Target Responsibilities

The VC Target is responsible for accepting the request from the OCB Target Wrapper and accomplishing the transfer. Since every request presented to the VC Target is intended for it, the VC Target should be designed to acknowledge *every* transfer. In simpler terms, the ACK response will often be a delayed version of the VAL. The VC Target should typically delay assertion of ACK until it has completed the transfer of the request; this forces the OCB Target Wrapper to hold the request information steady, thus eliminating the need for the VC Target to store the request.

The provider of the VC Target must provide a list of VCI configuration parameters for the VC, including such aspects as address/data bus widths, supported transfer widths, and timing data. These parameters allow the system integrator to configure the OCB Target Wrapper to meet the constraints of both the system and the VC Target.

5.1.5 VCI to VCI Conversions

VCI components with different parameters, as size, are not supposed to be connected together directly. A special wrapper has to be used in this case (See Figure 32). The wrapper is a VC, which takes care of buffering, and other logic needed to map different VCIs together. This situation happens, when using VCI for point-to-point communication. When connecting different VCIs through an OCB, the OCB wrappers take care of size conversions.

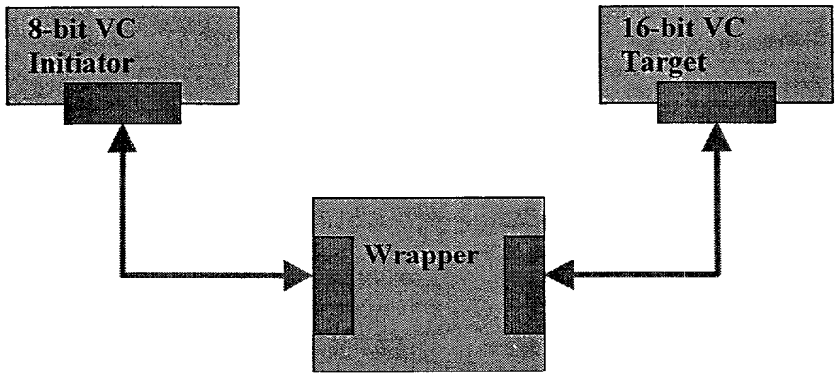

Figure 32. Interconnecting Different-Size VCI Components

5.2 VCI Parameters

These parameters control:

- Generation of the VC itself for a soft VC.
- Configuration signal/field tie-offs.

These parameters can also be used by EDA vendors to create tools for automating mixing & matching of VCs and Buses, and by IP providers to document the setup of the VC.

The default value of these parameters is 0 for all of the size/width type of parameters, and True for the binary type.

Independent of the implementation model abstraction level, language or format, these parameters must be documented. The implementation of the parameters in the models can be done with generics (e.g. in an HDL), setup or header files, etc. In case of a configurable hard VC, some of these parameters may be implemented with external pins that can be tied-off. 'True' means high signal value, and 'False' low signal value in the lists below.

5.2.1 Parameters Specific to PVCI

INITIATOR	-	True if VCI Initiator
ADDRSIZE	-	Address size, width of address bus in bits, range 0 to 31
CELLSIZE	-	Cell size, number of bytes in the data bus, value 1,2 or 4
FreeBE	-	True, if the VCI supports unrestricted combinations of byte enables. The restricted BE combinations, as defined in 3.4.3 must be supported by all VCI implementations.
BIGENDIAN	-	True, if component is big endian, False if little endian.
NOENDIAN	-	True, if component does not care of endianness (e.g. a memory). Overrides the BIGENDIAN parameter.
DefACK	-	ACK is allowed to be asserted even when VAL is low (deasserted)
RESETLEN	-	Number of clocks required of reset if NA or 0 use default (8 clocks)
ERRLEN	-	Number of error extension bits (defined as E), default is 0

5.2.2 Parameters Specific to BVCI

INITIATOR	-	True if VCI Initiator Initiator contains all signals and fields type MA and MI of table 5. Target contains all signals and fields of type MA of table 5.
CHAINING	-	True if VCI is capable of packet chaining Chaining VCI contains all signals and fields of type MC of table 5.
CELLSIZE	-	Cell size, number of bytes in the data bus
ADDRSIZE	-	Address size, width of address bus in bits
CLENSIZE	-	CLEN size, width of CLEN field in bits
BIGENDIAN	-	True for big endian, False for Little endian
NOENDIAN	-	True for VC which doesn't care of endianness
DefCMDACK	-	CMDACK is allowed to be asserted even when CMDVAL is low (deasserted)

DefRSPACK	-	Initiator is always ready to acknowledge RSPVAL (initiator is not allowed to insert wait state).
RESETLEN	-	Number of clocks required of reset if NA or 0 use default (8 clocks)
ERRLEN	-	Number of error extension bits (defined as E), default is 0

Default values for command extensions; for initiator the following parameters state that it will always send request of a certain behavior, as for target it only expects request of a certain behavior. The signal itself may or may not exist, but the behavior is known through these parameters, and the signals can be connected (tied off) accordingly.

Dcfixed	-	signal cfixed is tied to High/Low
Dconst	-	signal const is tied to High/Low
Dcontig	-	signal contig is tied to High/Low
Dwrap	-	signal wrap is tied to High/Low

5.3 Implementation Guidelines

Since the VCI is fully synchronous, and the wiring between the wrapper and a VCI component is supposedly very short, the VCI implementation should be very easy to design. A legal VCI component *must* sample all the signals at rising edge of the CLOCK. To create a legal VCI bus wrapper for an OCB that samples at the falling edge of the clock, buffering registers must be added.

The VCI standard should not pose any restrictions to making EDA tools that synthesize the bus wrapper automatically. This approach is preferred over manual wrapper design, since it reduces need for implementation verification, at least in theory.

In register transfer level implementation of the VCI, there are few restrictions: The flip-flops inferred must be rising-edge clocked and have active-low reset. The VCI standard does not pose any other electrical constraints to the physical implementation than those timing constraints given in signal definitions. The VCI component provider must document the physical implementation parameters and constraints, such as voltage swings. The guidelines given in VSIA Implementation/Verification DWG specifications and in On-Chip Bus DWG Attributes specification are to be followed. The manufacturing test constraints, or debug structures are not defined in the VCI standard. The guidelines given in VSIA Manufacturing Test DWG specifications are to be followed.

In a case of partly or fully combinatorial wrapper, the delay paths starting and ending at VCI component traverse all the way through the on-chip bus. It is the system integrator's responsibility to ensure the proper system timing in this case.

6. VCI Glossary of Terms

VC Interface	An OCB standard virtual component interface to communicate between a bus and/or virtual component, which is independent of any specific bus or VC protocol.
VCI Operation	A VC operation is a transport object consisting of a pair of packets, which are transferred in different directions e.g. a single request-response packet pair.
Cell	A cell is a grouping of one or more bytes. It contains a number of bytes that is characteristic to the natural width of the VCI implementation. It is the basic unit of information transferred across the VCI in one cycle.
Packet	A packet is transport object consisting of an atomic ordered set of cells transferred across the VCI.
Packet Chain	A packet chain is a non-atomic specialized transport object consisting of a set of logically connected packets transferred in the same direction across a VC Interface. The chain of packets is connected because no intervening packets are allowed on the same channel.
Signal	Synonymous to electrical wire or net.
Transaction	Generic term used for any pair of request and response transfer.
Transaction Layer	This deals with point to point transfers between blocks or virtual components. It does not define signal names, or clock-cycle protocols.
Initiator	A VC that sends request packets and receives response packets. It is the agent that initiates transactions e.g. DMA
Target	A VC that receives request packets and sends response packets. It is the agent that responds (with a positive acknowledgment by asserting) to a bus transaction initiated by a initiator – e.g. memory.
Bus Wrapper	Logic between the VCI and a bus.
VC Wrapper	Logic between an existing VC and the VCI.